UNIVERSIDADE ESTADUAL DE CAMPINAS

FACULDADE DE ENGENHARIA ELÉTRICA

OUTUBRO DE 1991

REPRESENTAÇÕES COMPUTACIONAIS AUXILIARES AO ENTENDIMENTO DE CONCEITOS DE PROGRAMAÇÃO

Este exemplar corresponde à redação final da tese defendida por Heloisa Vieira da Rocha Corrêa Silva aprovada pela Comissão Julgadora em 10 / 10 / 91.

Orientador

autor: Heloisa Vieira da Rocha Corrêa Silva
orientador: Prof. Dr. José Armando Valente

tese apresentada à Faculdade de Engenharia Elétrica, da Universidade Estadual de Campinas, como parte dos requisitos exigidos para obtenção do título de Doutor em Engenharia Elétrica

à Mariana e Daniel

AGRADECIMENTOS

Gostaria de expressar meus agradecimentos às pessoas e organizações que tornaram este trabalho possível. Em particular, gostaria de agradecer:

- ao Prof. Dr. José Armando Valente, pela orientação segura durante todo o desenvolvimento do trabalho.

- aos alunos Aloísio Soares e Ricardo Arruda, pela inestimável ajuda na implementação do sistema computacional.

- aos colegas e amigos do Núcleo de Informática Aplicada à Educação (NIED), da UNICAMP, pelo constante incentivo.

- ao Prof. Dr. Cláudio L. Luchesi, por ter viabilizado junto ao Departamento de Ciência da Computação (DCC), da UNICAMP, o início de meu programa de doutorado.

- à amiga Fernanda, pela paciência em ler grandes partes do meu trabalho para corrigir erros de linguagem.

- aos amigos Valtinho e Anamaria, pela torcida.

- à querida amiga Sílvia, pela presença constante e apoio nos momentos mais difíceis.

- à minha família pelo apoio constante.

- ao Michel, pelo carinho com que me acompanhou na fase final deste trabalho.

- ao CNPQ, que financiou por dois anos esta pesquisa.

- ao NIED, pelo ambiente e infraestrutura indispensáveis ao desenvolvimento de minha pesquisa.

- ao DCC, por ter favorecido através de dispensa parcial de carga didática, o desenvolvimento de meu programa de doutorado.

- à Faculdade de Engenharia Elétrica (FEE), da UNICAMP, em especial ao Prof. Dr. Márcio L. de Andrade Neto, pela liberdade dada ao desenvolvimento deste tema de pesquisa.

RESUMO

O objetivo deste trabalho foi o de explorar o uso de representações computacionais, em tempo real, como auxílio ao aprendizado de conceitos e processos abstratos. O domínio de aprendizado escolhido foi programação de computadores. Programação de computadores foi escolhido por existirem conceitos e processos tais como variáveis, fluxo de execução, recursão, iteração, etc., difíceis de serem assimilados através de meios convencionais. Considerando-se a linguagem Logo aliada a sua metodologia de aprendizado foi desenvolvido um sistema que ampliou o ambiente de programação, criando um ambiente onde são apresentadas representações alternativas de um programa em execução. O objetivo do sistema foi facilitar o desenvolvimento de um modelo mental adequado que guiasse toda interação do usuário com a linguagem de programação.

O projeto do sistema baseou-se na hipótese de que uma das dificuldades básicas para entender esses conceitos computacionais é a falta de visibilidade do processo de execução dos programas. Dada a opacidade do computador enquanto máquina e a natureza abstrata dos conceitos computacionais, deve-se utilizar outras representações caso se deseje tornar o processo de execução de alguma forma "visível" . E como o processo é dinâmico deve-se ter representações dinâmicas.

Os resultados mostraram que as pessoas realmente têm um modelo mental falho do funcionamento de processos computacionais, mais especificamente, de procedimentos recursivos. Isto não se restringe às pessoas não especialistas em computação. O uso do sistema, na maioria dos casos, mostrou-se útil como ferramenta auxiliar na correção e aquisição de modelos mentais adequados.

Os resultados obtidos neste trabalho no contexto de programação podem ser estendidos para outras áreas de conhecimento, como Física, Química, Biologia, etc... Nesses domínios existe uma série de conceitos abstratos como velocidade, equilíbrio, pressão, etc., também bastante difíceis de serem compreendidos. Certamente a compreensão destes conceitos pode ser facilitada com o uso de outras tipos de representações de fenômenos onde estão presentes estes conceitos.

Résumé

Le but de ce travail concerne le fait d'explorer l'usage de représentation computationelle em temps réel, comme une aide pour l'apprentissage des concepts et processus abstraits . Le domaine de apprentissage choisi a été la programmation d'ordinateur. La raison de ce choix tient à la difficulté par assimilation des concepts et processus, tels quels, variables, flux d'execution, recurrence, itération, etc., par de moyens conventionnels. Utilisant le langage de programmation Logo associé à sa mèthodologie d'apprentissage, on a developpé un système qui a élargi l'environnement de programmation, créant à la fois un autre environnement où sont présentées les rèpresentations alternatives d'un programme en train d'être exécuté. Le système a pour but faciliter le développement d'un modèle mental pour guider l'interaction entre l'utilisateur et le langage de programmation.

Le projet du système a été basé sur l'hypothèse que une des principales difficultés pour compreendre les concepts computationelles se rapporte à la manque de visibilité du processu d'exécution des programmes. Ainsi, en tenant compte de l'opacité de l'ordinateur en tant que machine, et le caractère abstrait des concepts computationelles, il faut utiliser d'autres rèpresentations dans le cas où l'on dèsire rendre le processus d'exécution plus visible. Comme le processus est dynamique, il faut donc avoir des représentations dynamiques.

Les résultats obtenus ont montré que les personnes possédent modèles prèsentant des fautes sur le fonctionnement des processus computationelles, surtout en ce qui concerne la recurrence. Cela s'applique même aux personnes specialisées en informatique. L'usage du système, dans la plupart des cas, s'est montré utile pour la correction et acquisition des modèles mentaux appropriés.

Les résultats obtenus dans ce travail dans le domaine de programmation peuvent être appliqués aussi dans autres domaines, comme par exemple, la Physique, la Chimie, la Biologie, etc., où existent plusieurs concepts abstraits - velocité, equilibré, prèssion, etc.- dont la compréhension est très dificile. Certainement on peut en faciliter la comprehension avec l'usage d'autres types de représentation des phénomenes où ces concepts sont présents.

ABSTRACT

The aim of this thesis was to explore the use of real time computational system of representations, as an aid to the learning of abstract concepts and processes. The chosen learning domain concerned computer programming. The choice was based on the fact that there are many concepts and processes, such as, variables, execution flow, recursion, loops, etc., which pose several difficulties to be assimilated through conventional means. The Logo programming language together with its learning methodology was utilized in order to develop a system that enlarged the programming environment, adding another environment where alternative representations of a program under execution are presented. The goal of the system was to facilitate the development of an adequate user mental model to guide the interaction between the user and the programming language.

The design of the system was based on the assumption that one of the basic difficulties to understand computational concepts is the lack of visibility of the processe that involve the execution of the programs. Give the opacity of the computer machinery and the abstract nature of the computational concepts, one should utilize representations so as to render the execution processe somewhat more "visible". These representations should be dynamic since the processe involved is dynamic itself.

The results showed that people possess misconceptions in their models of how work computational processes, more precisely, a recursive procedure. This fact applied to people other than non-specialists in computing. In most cases, the use of the system proved to be a useful tool in the acquisition and adjust of mental models.

The results obtained in this thesis in the programming concepts context, could be extended to another knowledge domain, such as, Physic, Chemistry, Biology, etc., where the comprehension of several abstract concepts - velocity, balance, pressure, friction, etc.- present difficulties. Their comprehension could certainly be facilitated by the use of other kinds of representation of the phenomena where these concepts are present.

INDICE

1.Introdução.....13

2. Ensino Convencional de Programação.....47

2.1 Introdução.....48

2.2 Princípios.....49

2.3 Apresentação dos Conceitos Básicos.....56

2.4 O Conceito de Recursão.....60

2.5 Representações Computacionais Estáticas.....69

2.5.1 Diagrama de Execução.....69

2.5.2 Cadeias de Execução.....72

2.5.3 Diagrama da Sequência de Chamadas.....74

2.5.4 Tabela de Rastreio.....77

2.6 Ambientes de Depuração.....80

2.7 Conclusão.....84

3. Sistemas Computacionais no Ensino de Programação.....88

3.1 Introdução.....89

3.2 Tutores/ Instrutores.....91

3.3 Depuradores Inteligentes ("Bug Finders").....108

3.4 Ambientes de Suporte.....114
3.4.1 Bip.....114
3.4.2 Bridge.....115
3.4.3 Tinker.....117
3.4.4 Boxer.....121
3.5 Conclusão.....127

4. Logo: Linguagem e Metodologia.....130
4.1 Introdução.....131
4.2 Logo como Linguagem de Programação.....131
4.2.1 Aspectos Gerais.....132
4.2.2 Descrição da Linguagem Logo.....141
4.3 Metodologia Logo.....165
4.4 Problemas Conceituais.....172
4.4.1 Procedimentos e Fluxo de Execução.....172
4.4.2 Variáveis.....174
4.4.3 Recursão.....177
4.5 Conclusão.....181

5. Modelos Mentais.....183

5.1 Introdução.....184

5.2 Caracterização de Modelos Mentais.....187

5.3 Natureza dos Modelos Mentais.....191

5.4 Modelo Substituto ("Surrogate Model").....194

5.5 Modelos Mentais e Conceitos de Programação.....201

5.6 Conclusão.....212

6. Objetivo e Metodologia.....214

6.1 Introdução.....215

6.2 Objetivo.....215

6.3 O que Originou a Escolha deste Tema.....217

6.4 A Escolha da Linguagem Logo.....219

6.5 Metodologia.....223

6.5.1 Método de Definição do Sistema.....224

6.5.2 Método de Observação.....226

7. Descrição do Sistema Implementado.....230

7.1 Introdução.....231

7.2 Apresentação do Sistema.....232

7.3 Exemplo de Uso.....248

7.4 Descrição da Implementação.....265

7.4.1 Aspectos Gerais.....265

7.4.2 Conjunto de Intruções Implementado.....266

7.4.3 Estrutura de Dados.....268

8. Uso do Sistema para Diagnosticar e Remediar o Conceito de Recursão.....278

8.1 Introdução.....279

8.2 Aspectos Gerais da Oficina de Trabalho.....281

8.3 Primeiras Atividades: Diagnóstico.....284

8.3.1 Descrição da Atividade.....284

8.3.2 Justificativa e Hipóteses Iniciais.....286

8.3.3 Resultados Obtidos.....288

8.4 Fase de Uso do Sistema.....302

8.4.1 Primeiro Uso.....302

8.4.2 Primeiros Resultados do Uso.....303
8.5 Mudança de Paradigma: O Procedimento como Operação.....311
8.5.1 O Problema do Inverte.....317
8.5.2 O Problema da Palíndrome.....322
8.5.3 Atividades Finais.....325
8.6 Conclusão.....330

9. Uso do Sistema no Aprendizado de Recursão no Processamento Simbólico.....335
9.1 Introdução.....336
9.2 Primeira Fase de Diagnóstico.....340
9.3 Apresentação do Sistema.....348
9.4 Segunda Fase de Diagnóstico.....351
9.4.1 Descrição dos Problemas.....351
9.4.2 Resultados Obtidos.....352
9.5 Introdução do Comando Envie.....362
9.6 Terceira Fase de Diagnóstico.....366
9.7 Conclusão.....378

10. Quanto os Especialistas de Computação Dominam o Conceito de Recursão: Um Estudo de Caso.....380

10.1 Introdução.....381

10.2 Primeira Fase : Diagnóstico.....383

10.2.1 Pascal.....383

10.2.2 Logo Gráfico.....388

10.2.3 Logo Listas.....393

10.3 Segunda Fase : Uso do Sistema.....397

10.4 Conclusão.....405

11. Discussão e Conclusões.....409

11.1 Introdução.....410

11.2 Aprender a Programar é difícil.....413

11.3 Modelo Mental sobre Computador e Conceitos Computacionais é Falho.....422

11.4 Recursos de Ensino não são Adequados.....424

11.5 O que foi feito e como ajudou à compreensão dos modelos que as pessoas tem de conceitos computacionais.....429

11.6 Conclusão.....435

Bibliografia.....441

Apêndice 454

Capítulo 1

Introdução

Num futuro próximo a maior parte dos usuários de computadores serão pessoas não especializadas em computação e para as quais o computador será uma ferramenta útil, senão indispensável, a seus interesses e atividades profissionais. Com esta perspectiva, observa-se hoje uma grande disseminação de cursos particulares de computação em todas as áreas e dirigidos a todo tipo de público: crianças, adolescentes, executivos e uma ampla gama de profissionais.

Dentro do âmbito da universidade, têm sido oferecidos cursos introdutórios de programação a praticamente todas as carreiras. Estes cursos atendem solicitação dos responsáveis pelos diversos bacharelados, que por sua vez atendem à solicitação de alunos que sentem necessidade de uma introdução básica em computação.

Portanto existe uma evidente necessidade e interesse em computação. Entretanto os cursos introdutórios de programação, oferecidos na universidade para alunos das áreas de exatas e humanas, iniciam suas turmas com uma média de 60 alunos e em poucos meses estão com 20. Obtém-se portanto uma média de 60% de desistência, o que sugere pouco interesse por esta disciplina, o que sem dúvida é uma contradição.

Algumas hipóteses são levantadas com relação a este fracasso. A mais comum é que talvez não seja via programação que esta gama de novos usuários de computadores deva ser introduzida à computação. A partir desta hipótese surgem alternativas para esta introdução como a utilização de aplicativos tais como planilhas, gerenciadores de dados, etc. Mas o uso destas ferramentas é extremamente limitado se a pessoa que vai utilizá-las não tiver nenhuma conceituação de variáveis, sequencialidade de execução de instruções, iteração, etc., ou seja, alguns conceitos básicos de Ciência da Computação. Além disso a maior parte destas ferramentas apresenta dois modos de operação: o interativo ou direto e o programado, e geralmente é via o modo programado que são feitas a maior parte das aplicações interessantes. Se o uso destas ferramentas ficar restrito ao modo direto, com utilizações extremamente simples, o aluno não adquirirá nenhum conceito computacional e também não ganhará conhecimento sobre o potencial dos aplicativos.

Portanto, acho inquestionável o fato de que a melhor introdução à computação seja via programação, mesmo considerando usuários que não necessariamente escreverão programas. Através da programação torna-se claro o funcionamento do computador e de suas capacidades, e esse conhecimento é essencial para qualquer uso que se venha a fazer dele.

Precisamos então questionar o que está errado com o ensino de programação. O que pode ser feito para procurar reverter esta situação?

A meu ver, alguns pontos do ensino de programação precisam ser repensados:

1- a metodologia de ensino que não diferencia os alunos, considerando todos como futuros profissionais de computação.

2- o conteúdo apresentado onde é dispendido grande tempo no ensino da semântica e sintaxe da linguagem de programação utilizada.

3- a não atenção aos diferentes estilos cognitivos dos alunos que de certa maneira reflete na atividade de programação.

4- a falta de recursos que facilitem a compreensão de conceitos computacionais, auxiliando as pessoas a construírem um modelo mental adequado do funcionamento do computador. Este é o foco principal deste trabalho.

O ensino de programação tem sido uma área muito bem estabelecida, determinado por um paradigma denominado programação estruturada, aliado ao uso de linguagens imperativas ou procedurais. De acordo com este paradigma o desenvolvimento de um programa segue um planejamento dirigido do problema para a produção de um programa, estabelecendo uma estrutura fechada de desenvolvimento, que compreende os seguintes passos:

1 -especificar um plano detalhado que deverá ser implementado. Isto geralmente é feito decompondo-se a tarefa inicial em sub-tarefas mais simples de modo que a solução das sub-tarefas, em uma certa ordem, leva à solução do problema original.

2 - mapear o plano obtido dentro de construções de uma linguagem de programação.

3 - depurar o programa obtido.

Os cursos de programação enfatizam a necessidade de seguir esta sequência de passos para obter um programa. A abordagem está também amarrada às linguagens de programação utilizadas que forçam a estruturação do programa, como por exemplo, a linguagem Pascal.

O passo principal, nesta abordagem, é a elaboração do algoritmo, e nos cursos de uma maneira geral, é passada a idéia de que aprender a programar significa aprender a elaborar algoritmos.

De maneira geral, um curso introdutório de programação dentro da universidade começa pela elaboração de algoritmos, ou seja, dado um problema descrever sua solução através de uma sequência de passos finita. Primeiro são elaborados algoritmos para problemas cotidianos, como o uso de um "orelhão", trocar o pneu de um carro, etc. Nesta fase o objetivo é mostrar que de modo geral as pessoas convivem com algoritmos em suas atividades diárias e que a dificuldade maior consiste em como descrevê-los de uma maneira formal e inteligível para ser executado por outras pessoas.

É enfatizada a necessidade de especificar qual o tipo de conhecimento da pessoa a quem se destina o algoritmo para saber quais são as ações primitivas que podem aparecer no algoritmo. Por exemplo, no algoritmo do uso do telefone, pode-se ter uma ação como "disque o número desejado" se a pessoa sabe discar, caso contrário, deve-se decompor esta operação em passos mais simples que ensinem a discar um número. Os algoritmos dos alunos

são os mais diversos possíveis e geralmente eles não sentem muita dificuldade em escrevê-los, provavelmente pela não rigidez no modo de expressar o algoritmo e a facilidade de inferir as ações, tendo em vista que o algoritmo se destina a outra pessoa.

Em seguida, é apresentado o computador. São definidas quais as ações que podem aparecer em um algoritmo a ser implementado no computador. Estas ações são descritas através de uma pseudo-linguagem muito próxima da linguagem de programação a ser utilizada no curso. Um dos objetivos do uso desta pseudo-linguagem é o de criar uma certa independência entre o que é um algoritmo e o que é o programa final a ser executado pelo computador, mostrando que uma vez pronto o algoritmo escrevê-lo em uma linguagem de programação não passa de uma mera tradução. Outro objetivo é o de facilitar a escrita de algoritmos destinados a serem resolvidos pelo computador, utilizando para isso uma linguagem mais natural, sem grandes restrições sintáticas e em Português.

O que se deseja é que os alunos persistam nesta prática de escrever programas, seguindo estas duas fases "distintas": primeiro obter um pseudo-código e depois o programa final, e que

com isso ganhem autonomia quanto à programar em qualquer linguagem de programação. Mas o que ocorre é o abandono pelos alunos desta pseudo-linguagem tão logo lhes é apresentada a linguagem de programação. Ao tentar forçar a idéia da necessidade do algoritmo o que se observa é um grande questionamento por parte dos alunos, sendo frequentes afirmações do tipo "eu só consigo fazer um programa na frente do computador, pois assim eu vou vendo o que acontece a cada instrução que coloco e não preciso ficar imaginando o que vai acontecer", ou então, "a gente não pode fazer o algoritmo depois do programa estar funcionando?", e também, "não sei para que fazer um algoritmo certo, se eu posso ir consertando a idéia na frente do computador" e "fazer programa eu sei, o que preciso estudar é algoritmo".

Estas afirmações nos levam a questionar o paradigma da programação estruturada. Será que ele não está sendo uma "camisa de força", criando obstáculos ao aprendizado de programação? Como consequência desta questão, será que as linguagens utilizadas em cursos iniciais são as mais adequadas?

Deve-se considerar que programação estruturada é uma proposta de engenheiros, que foram os primeiros profissionais envolvidos com programação de computadores, e portanto nada mais natural que a ênfase ao planejamento e estrutura. Entretanto, será que todas as pessoas que querem aprender programação devem necessariamente se adequar ao estilo "engenheiro"? Não questiono a eficácia da programação estruturada e sim o senso comum de que esta é a melhor maneira de programar.

Além disso, dentro do contexto de programação, outros paradigmas de programação vem ganhando popularidade, dada a mudança da natureza de problemas que são tratados computacionalmente: o paradigma funcional, o orientado para objeto e o lógico.

No paradigma funcional os programas são definições de funções e a execução consiste na avaliação de aplicações de funções. As linguagens funcionais tentam imitar funções matemáticas o máximo possível. Isto resulta em uma abordagem de solução de problemas que é fundamentalmente diferente dos métodos utilizados com linguagens procedurais ou imperativas. Muitas linguagens de programação funcional tem sido desenvolvidas. A mais antiga e

mais amplamente utilizada é o LISP, originariamente projetada para as primeiras aplicações de Inteligência Artificial. Uma linguagem funcional é aquela na qual o meio principal de fazer computações é através da aplicação de funções com determinados parâmetros. Programação pode ser feita em uma linguagem funcional sem variáveis, sem comandos de atribuição e sem iteração.

Dentro da área de metodologia de programação, o projeto orientado para objeto tem tido um importante impacto. Nesta abordagem é enfatizado o projeto de dados, sendo portanto uma abordagem oposta à tradicional que, como foi visto, focaliza o processo e a sua implementação através de procedimentos. A linguagem de programação mais utilizada neste paradigma é a Smalltalk, que é realmente uma linguagem pequena, com uma sintaxe simples e muito regular, mas implementada em um ambiente de programação extremamente poderoso.

O paradigma de programação lógica é o que conduz à expressar programas na forma de lógica simbólica e usar o processo de inferência lógica para produzir os resultados. A linguagem PROLOG é a mais amplamente utilizada e com uma crescente popularidade.

As linguagens de programação lógica são também chamadas de declarativas pois os programas lógicos consistem de declarações ao invés de atribuições e comandos de controle de fluxo.

Os programas escritos sob estes três paradigmas não estabelecem exatamente como o resultado deve ser calculado, mas descrevem a forma do resultado. Onde ficam os algoritmos tão amplamente exigidos e tão fundamentais para a escrita de um programa?

O movimento de programação estruturada teve lugar no início dos anos 70, e alterou tanto o processo de programação como o projeto de linguagens. O resultado destes esforços tem sido consolidado sob o título de "Engenharia de Software".

Existem duas visões conflitantes em Ciência da Computação que se pode denominar de visão da Engenharia de Software e da visão de Inteligência Artificial. Na primeira o processo de desenvolvimento de um programa é visto como começando com uma idéia clara e bem definida do que o programa deve fazer. Esta idéia é escrita na forma de uma especificação do programa que será a base da programação. A partir deste ponto o programa é

desenvolvido da maneira "top-down". Nenhum procedimento é escrito sem que esteja claro como será encaixado dentro do programa como um todo.

Em contrapartida, na visão de Inteligência Artificial, o desenvolvimento de um programa é um processo mais complicado envolvendo tanto o processo "top-down" como o "bottom-up". É apropriado portanto iniciar escrevendo fragmentos de programa que tratam sub-tarefas já entendidas e então tentar verificar como estes pedaços podem ser colocados juntos para completar o projeto. O desenvolvimento não é uma linha reta a partir de uma especificação abstrata para procedimentos concretos; é um "zig-zag" dentro do qual o programador tem uma idéia, tenta, e então usa os resultados para pensar mais sobre outras idéias.

Esta concepção do processo de programação tem sido muito ressaltada dada a mudança na natureza das aplicações computacionais atuais. Não mais se parte de um problema totalmente entendido que permita uma completa especificação. Programação exploratória [Sheil, 1984] é mais adequada nestes casos, pois possibilita um design mais flexível do que o que é conseguido através de metodologias convencionais de programação.

Naturalmente aliado a esta concepção tem-se toda uma ênfase no desenvolviemnto de ambientes e ferramentas interativas [Barstow, 1984] que viabilizem esta forma de programação.

Esta diferenciação entre visões do processo de programação, nos remete novamente à afirmação do aluno de que "não sei para que fazer um algoritmo certo, se eu posso ir consertando a idéia na frente do computador". Esta liberdade no modo de escrever um programa em muito foi auxiliada com o advento dos microcomputadores. Programação estruturada foi criada a partir do uso de computadores "main frames", onde não existia a interação do usuário e o computador. Era preciso fazer tudo certinho, pois as chances de executar o programa eram poucas. Dessa forma era eliminada a tentativa e erro. O aluno querer criar o programa em frente ao computador é algo novo, que só foi possível com o advendo do micro. Entretanto, o ensino de programação não incorporou esta mudança de hábitos e suas consequências.

Outro aspecto do ensino de programação a ser repensado é o grande tempo que é dispendido nos cursos introdutórios ensinando à sintaxe e semântica da linguagem de programação. Perguntei a um professor como estava indo seu curso e ele me respondeu "está

ótimo, pois já consegui dar o comando Repeat, While, For, Atribuição e comecei com Arrays, e os alunos parecem estar entendendo tudo". Não houve menção à qualidade dos problemas que estavam sendo resolvidos e nenhuma menção aos conceitos trabalhados.

Isto leva a que os alunos, de modo geral, tenham um conhecimento das construções da linguagem mas não consigam "colocar as peças juntas" coordenando componentes de um programa. Não são ressaltados aspectos relativos ao processo de solução de problemas que norteiam à atividade de programação. A ênfase na sintaxe e semântica leva a focalizar o programa como resultado do processo de programação. Por que aprender onde por "um ponto e vírgula" em Pascal? Isso leva a um aumento na habilidade de resolver problemas?

Com os cursos nesta forma, estamos tendo um fracasso no ensino de programação. Para as pessoas que não questionam a maneira de ensinar, resta a conclusão confortável de que "infelizmente programação não é para todos". Já para uma razoável comunidade científica, surge questionamentos sobre como ensinar programação.

são pesquisadores preocupados em entender este processo de aprender a programar, detectando falhas e dificuldades deste aprendizado e sugerindo alternativas, de modo a facilitar o aprendizado de programação.

Como alternativas ao convencional ensino "giz e quadro-negro", surgem algumas propostas, que utilizam o computador como auxiliar ao ensino/aprendizado de programação. Estas propostas podem ser enquadradas em três categorias bastante amplas de acordo com os objetivos educacionais que propõem: tutores, depuradores inteligentes e ambientes de suporte.

Os tutores são sistemas computacionais que atuam no sentido de tomar decisões sobre como e em que ordem conceitos de programação devem ser introduzidos, e acompanhar o progresso do estudante no domínio destes conceitos.

Estes sistemas apresentam uma rigidez na abordagem de como deve ser o ensino de programação, desde a sequência pré-definida de conceitos a serem ensinados até a pré-concepção do aluno a ser ensinado. Tem-se portanto um esquema fechado, onde o aluno ou se molda à concepção do que o sistema propõe como programação ou

então não programa. Este tipo de sistema se adequa mais ao treinamento que ao desenvolvimento de novos conceitos.

Os sistemas denominados depuradores na sua maioria concentram-se sobre erros de lógica, ao invés de sintáticos. Estes sistemas adotam diferentes métodos para isolar e detectar "bugs": casamento com uma resposta pré-definida pelo sistema, análise se o programa corresponde a um plano definido pelo aluno, etc. Com o auxílio do sistema e diálogos com o usuário é possível detectar se o que era esperado pelo programa corresponde ao que está efetivamente acontecendo.

De modo geral, os depuradores inteligentes coexistem com outros métodos de ensino e devem se adequar ao estilo do ensino que é efetuado. Mesmo assim o nível de "inteligência" de tais sistemas é muito sutil, pois dificilmente se consegue verificar, dado um erro, qual era a intenção da pessoa que escreveu o programa. Mesmo a nível de erros de sintaxe, que é um contexto bem mais fechado e definido que o nível de lógica de concepção do programa, não se consegue escrever um compilador ou interpretador que corrija os erros, pois em muitos casos a correção mais óbvia não corresponde à intenção do programador, e pode vir a inserir

mais erros. Analogamente, mesmo que os depuradores não se proponham a corrigir erros, podem com suas sugestões, vir a criar mais confusão que ajudar a depuração da solução.

Ambientes de suporte é a categoria mais difusa de sistemas e consiste basicamente de sistemas que provêem um meio de tornar a interação do estudante com o sistema computacional mais efetiva e compreensível. Nestes sistemas encontram-se ferramentas integradas tais como programas de rastreio, editores de alto nível, linguagens de programação que visam facilitar o processo de programação, etc. De modo geral são toda espécie de ferramenta computacional construída objetivando aprendizes e usuários não especialistas em computação. Torna-se portanto difícil caracterizar tais sistemas, a não ser descrevendo alguns deles, o que será feito no Capítulo 3 deste trabalho, juntamente com a descrição de implementações de sistemas das outras duas categorias.

Todo o questionamento com relação ao paradigma da programação estruturada e a maneira como vem se efetuando o ensino de programação, vem ganhando muita força devido à amplitude crescente do escopo de usuários e a proximidade com a máquina,

graças à utilização de micro-computadores. Atualmente existe uma grande preocupação com relação a facilitar a interação com o computador. Um exemplo são os editores de texto, que sem dúvida são os sistemas mais utilizados.

Observa-se que as pessoas que utilizam os sistemas de editores de texto, estão mudando sua maneira de escrever. A facilidade de interação com estes sistemas as desobrigam de escrever no papel antes de datilografar um texto. As idéias se desenvolvem juntamente com o texto. Isto é conseguido graças à facilidade de comunicação com o computador que tais sistemas oferecem. Isto em muito está relacionado com o que já foi discutido no início deste capítulo, sobre a liberdade no estilo de escrever programas; o que está acontecendo hoje com a criação de textos pelo autor a medida que ele vai datilografando, deve acontecer com o processo de programar. E o que é mais importante, todo o suporte computacional para programação deveria caminhar nessa direção.

Existem linguagens de programação que facilitam esta comunicação e consequentemente facilitam o aprendizado de programação através do uso.

Um exemplo é o subconjunto da linguagem Logo que controla a Tartaruga, onde através de um pequeno número de ações bem definidas e claras, resultados interessantes são obtidos rapidamente por principiantes escrevendo programas simples. Com um conjunto de dez instruções mais a estrutura de procedimentos, é possivel um ambiente onde efeitos gratificantes são obtidos nos primeiros contatos com o computador. Com Logo não existe etapa anterior ao uso do computador. O aluno é desde o início de seu aprendizado colocado em contato com o computador, desenvolvendo programas. Parte-se do princípio de que é fazendo e pensando sobre o que se faz que se efetua o aprendizado de conceitos.

O trabalho com Logo leva a uma outra abordagem do ensino de programação. No Logo é dado mais ênfase no processo de construção do programa do que no programa em si. A preocupação fundamental é entender o pensamento de quem programa. Isto é feito observando-se como as pessoas fazem e depuram seus programas quando interagem com um ambiente que propicia uma atuação livre de acordo com seus estilos próprios de resolver problemas.

Com isto observa-se diferentes estilos de resolução de um problema: desde o rigorosamente planejador até o absolutamente

empírico e explorador. Portanto sem uma "camisa de força", dando-se liberdade ao estilo individual e tendo-se ambiente adequado para que isto ocorra, pode-se afirmar que todas as pessoas programam. Para tornar mais clara esta idéia vou relatar um exemplo descrito por Turkle em [Turkle, 1984].

Em uma escola que possuía diversos microcomputadores e que trabalhava com a linguagem Logo, numa certa época do ano, o projeto de interesse dos alunos era o desenvolvimento de cenas espaciais. A versão da linguagem Logo que utilizavam era relativamente poderosa. Permitia a utilização de trinta e dois objetos computacionais denominados "sprites" que aparecem na tela somente quando solicitados. Cada "sprite" possui um número, pode-se dar a ele uma cor e uma forma e ele aparece na tela com a cor e a forma definidas: um caminhão amarelo, uma bola azul, um avião verde, etc. Existem formas pré-definidas mas pode-se editar qualquer forma desejada e associá-la a um "sprite". Pode-se movimentar o "sprite" atribuindo-lhe uma direção e uma velocidade e colocando-o em movimento. As cenas espaciais faziam amplo uso de "sprites" e não se podia dizer, observando-se somente o resultado final, como tinham sido feitas.

Geraldo, um menino da quarta série, tinha reputação de ótimo programador. Ele era meticuloso em seus hábitos de estudo e realizava um trabalho muito bom em todas as disciplinas, não sendo portanto, para seus professores, uma surpresa que ele viesse a se destacar em programação. Ele falava muito rápido e quando se referia a seus programas o fazia mais rápido ainda, tendendo a um monólogo. Escrevia rapidamente as linhas do programa sem precisar observar o código a medida que ia aparecendo na tela. Transmitia a sensação que falava diretamente com uma pessoa que estava dentro do computador e dizia "quando programo me coloco no lugar do "sprite" e começo a fazer coisas".

Geraldo foi um dos primeiros autores de uma cena espacial e a fez como fazia a maioria de suas coisas, por meio de um plano. O programa foi concebido de forma global e depois dividido em partes. Em termos computacionais esta técnica é denominada "top-down", "divide-and-conquer", e considerada como um bom estilo de programação respeitando o paradigma de programação estruturada. Reconhecidamente, Geraldo se encaixava no estereótipo de "pessoa de computação", ou seja, de alguém que vai ser bom com máquinas, bom em ciência, alguém organizado.

Ramiro era um menino de tipo completamente diferente. Da mesma forma que Geraldo era preciso em todas as suas ações, Ramiro era sonhador e impressionista. Ele também preparava uma cena espacial, mas o modo como a fez não se parecia em nada com o estilo de Geraldo. Para Geraldo não interessava em demasiado a forma de sua nave espacial, o importante era que o sistema funcionasse como um todo. Já para Ramiro interessava mais a estética de seus gráficos. Dedicou muito tempo elaborando os detalhes da nave e dizia "quando trabalho penso que sou o homem, que estou dentro da cápsula", o que era muito diferente de Geraldo que se imaginava como o "sprite", um objeto computacional abstrato.

Depois de muitas idas e vindas e consultas a opinião dos amigos, Ramiro conseguiu uma nave com uma bola de fogo embaixo, e partiu para colocá-la em movimento. É muito fácil colocar os "sprites" em movimento, bastando dar-lhes uma posição inicial, uma velocidade e direção. Sem um plano anterior, Ramiro colocou os "sprites" em movimento não levando em conta que sua nave e o respectivo fogo eram "sprites" independentes e que portanto seus movimentos deveriam ser coordenados. O que aconteceu então foi que a cápsula se afastou do fogo. Quando cometia um erro, Geraldo

se irritava e chamava a si mesmo de estúpido, e corria corrigir sua falha técnica. Ramiro, pelo contrário, ao corrigir seu erro, explorava o sistema, descobrindo novos efeitos especiais. Seu erro de movimentação o fez mudar de idéia, achando mais interessante que a cápsula se fosse e o fogo ficasse flutuando entre as estrelas. Ramiro sabia escrever programas mas de uma maneira nada sistemática permitindo que seus planos fossem modificados a medida que o programa ia se concretizando. Mas ao final da semana ele também havia programado uma cena espacial.

A observação destes dois meninos trabalhando com o computador, mostrou duas pessoas bastante diferentes e que obtiveram êxito na mesma coisa. Pode-se dizer que Ramiro não só obteve êxito como, da mesma forma que Geraldo, aprendeu muito sobre programação, sobre manipulação de ângulos, coordenadas, formas e ritmos. Obtiveram portanto êxitos iguais mas trabalharam de forma bastante diferente e é a isto que se denomina de estilo.

Nem todos os sistemas computacionais ou linguagens de programação oferecem um material suficientemente flexível para que se possa expressar diferenças de estilo e isto não significa que sejam menos adequados computacionalmente, mas sem dúvida

dificultam a utilização ao não permitirem diferentes modos de uso.

Mas para ensinar/aprender programação não basta uma linguagem facilitadora, pois muitos conceitos precisam ser amplamente entendidos para poderem ser utilizados corretamente. Por exemplo, do trabalho com Logo observa-se que as pessoas utilizam procedimentos com parâmetros e não conseguem trabalhar com variáveis dentro de uma atribuição. Apesar de utilizarem variáveis como parâmetros, não possuem a conceituação do que é uma variável numa linguagem de programação, para que possam generalizar sua utilização. Quanto à estrutura de procedimento e sub-procedimento, verifica-se que as pessoas utilizam a estrutura mas perdem-se ao tentar acompanhar o fluxo de execução de um programa. Fica claro que a simples utilização não implica no entendimento do conceito. Se não for observado este aspecto o quanto antes, não se obtem progresso em programação de modo a desenvolver atividades que envolvam conceitos computacionais mais sofisticados.

As falhas de conceituação, no trabalho com Logo, são sentidas ao se avançar da parte gráfica para a parte de manipulação

simbólica. Ocorrem sérias barreiras, pois o nível de abstração e a complexidade das ações crescem consideravelmente. Além disso, muda-se o paradigma de programação, do procedural para o funcional e portanto, uma programação em outro nível. Não é mais possível acompanhar a execução do programa o que de alguma forma é conseguido na parte gráfica, pois através da execução de programas que fazem desenhos fica espelhado o comportamento do programa passo-a-passo. Já no processamento simbólico, o que se obtem é simplesmente o resultado, na forma de listas, sentenças, palavras ou números. Torna-se difícil entender como o computador chegou ao resultado, ou não, pois não existem mais maneiras diretas de acompanhar a execução do programa.

Qual a dificuldade no entendimento destes conceitos?

Estes conceitos estão diretamente relacionados ao funcionamento do computador, que no caso é definido pela linguagem, e portanto não é visível. Neste aspecto o computador é uma máquina extremamente opaca, pois se abrirmos um computador e observarmos o que acontece quando um programa está sendo executado, não vamos ver absolutamente nada. A maneira que temos de tentar explicar é através de representações, que de alguma

forma espelhem o funcionamento. Permitindo-se esta "visibilidade" do processo de execução, o usuário obtem um melhor "feedback" de sua interação, o que poderá auxiliar no processo evolutivo de construção de seu modelo mental do funcionamento do computador.

Em cursos formais de programação para facilitar o entendimento de conceitos tais como variáveis e seu escopo de validade, super e sub-procedimentos, recursão, iteração, etc. usa-se representações que procuram espelhar a execução de um programa. Existe na literatura específica de programação diversas formas de representação: diagramas de execução, árvores de ativação, etc. Através destas representações formaliza-se um modo de verificar a execução de um programa em seus aspectos mais relevantes. Por serem representações estáticas, observá-las já construídas nada significa, o aluno deve construir a representação para poder assimilar o dinamismo do processo que está sendo representado. Mas se o aluno não entende como funcionam as construções da linguagem ele não consegue fazer a representação, pois para construí-las é necessário que o aluno já saiba como o programa vai ser executado. Na verdade estas representações são mais utilizadas como ferramentas de depuração,

"testes de mesa estruturados", que como auxiliares ao entendimento do funcionamento do programa.

Existem portanto conceitos abstratos essenciais ao aprendizado de programação. São conceitos que não podem ser adquiridos via observações concretas e estáticas devido a opacidade do computador como máquina dinâmica. Baseada na hipótese de que o fornecimento de representações alternativas auxiliam o entendimento de conceitos com estas características e, que em se tratando de conceitos computacionais o dianamismo das representações é essencial, defini meu trabalho de pesquisa que como um todo, subdivide-se em três áreas de estudo:

a) estudo de como pessoas entendem conceitos computacionais;

b) desenvolvimento de sistemas computacionais, de tempo real, que forneçam representações alternativas dos conceitos computacionais;

c) análise experimental dos resultados obtidos do uso do sistema desenvolvido, de modo a poder verificar se realmente auxiliam o entendimento, sob o ponto de vista da construção de

um modelo mental adequado do funcionamento do computador.

Desenvolvi um sistema que fornece diversas representações da execução de um programa escrito em Logo não gráfico. Um dos aspectos principais do sistema é a multiplicidade de representações, consequência das observações de pessoas usando computadores onde constata-se, como já foi dito, uma diversidade de estilos de aprendizagem e portanto, qualquer material de aprendizagem deve possuir diversidade ao invés de uniformidade. Outro aspecto é a característica dinâmica das representações, ou seja, elas são fornecidas em tempo real, ao mesmo tempo em que é executado o procedimento.

O sistema não tem como objetivo ensinar os conceitos e sim oferecer uma alternativa que facilite seu entendimento, é o "fazer ver para melhor entender".

O processo de escolha das representações que foram implementadas foi, pode-se dizer, "ad hoc", baseado em experiências anteriores de ensino de programação. Defini um conjunto de representações baseadas nas representações utilizadas estaticamente, adicionando primordialmente a característica de

tempo real. Os conceitos computacionais considerados no desenvolvimento do sistema foram fluxo de execução, variáveis e recursão. Acredito serem estes os conceitos fundamentais de programação e todos os demais, como iteração, procedimentos, etc., podem ser tratados em função deles.

O sistema possui dois subconjuntos básicos de representações que podem ser denominados de locais e globais. O subconjunto de representações locais tem como objetivo o de mostrar localmente num certo ponto de execução o que está acontecendo e o que o computador "conhece" à respeito do programa. Perde-se nesta representação a história passada da execução, ou seja, não se tem claro todo o processo que conduziu àquele momento da execução. Por acreditar que esta visão é relevante para o entendimento dos conceitos abordados, principalmente com relação a procedimentos recursivos e programas com muitos níveis de execução, definiu-se uma outra representação que preserve a história de execução do programa. Esta outra representação é a que denomino de global.

A verificação do sistema foi feita com um pequeno grupo de usuários. A primeira etapa de testes foi feita com um grupo de 15 pessoas durante um curso de aperfeiçoamento em programação Logo.

O curso tinha a duração de 16 horas e trabalhou-se exclusivamente com a parte de manipulação simbólica do Logo. Os alunos foram selecionados de um grupo de pessoas que trabalham com Informática e Educação, que já sabiam programar em Logo e para os quais programação é prática diária.

Depois disso o sistema foi utilizado, de modo bastante informal, por bolsistas do Núcleo de Informática Aplicada à Educação, alunos do curso de Ciência da Computação da UNICAMP, que já sabiam programar em Logo.

Deste uso preliminar do sistema observou-se que falhas conceituais não estavam restritas à profissionais de outras áreas que não computação. Dificuldades de entendimento dos conceitos básicos de programação existem inclusive para pessoas cujo perfeito entendimento é essencial na vida profissional, como o caso dos bolsistas, alunos de terceiro ano do curso de Ciência da Computação da UNICAMP. O inusitado foi o interesse gerado pelo sistema junto a estes usuários, sendo que ele foi originalmente concebido como dirigido à principiantes.

A partir destas observações iniciais fiz junto à alunos do curso de Computação um breve levantamento com relação ao entendimento de conceitos computacionais, mais especificamente sobre recursão. O levantamento baseou-se na solução de alguns problemas e na predição do resultado de outros. E novamente falhas sensíveis de conceituação puderam ser observadas. Utilizei o sistema rapidamente com alguns destes alunos, somente com o objetivo de verificar o interesse e motivação no uso, pois os problemas trabalhados foram extremamente simples dado que estes alunos não conheciam a linguagem Logo.

Utilizei o sistema mais longamente em dois outros cursos de aprofundamento, sendo um de 40 horas e outro de 20, envolvendo no total cerca de 20 profissionais ligados à area de Informática e Educação. Finalmente fiz um uso não muito prolongado mas de muito bom aproveitamento junto a um aluno de pós-graduação em Computação que conhecia a linguagem Logo.

Resultados interessantes foram obtidos melhorando em muito meu conhecimento de como as pessoas entendem conceitos computacionais e a descoberta da não existência de relação entre o utilizar e compreender.

Também pude concluir que efetivamente o entendimento de muitos dos conceitos de programação está diretamente relacionado com o entendimento de como efetivamente o computador processa um programa. Para tanto representações deste funcionamento em muito auxiliam e a característica de serem representações computacionais é importante dado os aspectos de tempo real e interatividade.

Certamente os resultados obtidos neste trabalho com o uso de representações computacionais alternativas de processos e conceitos abstratos no contexto de programação podem ser estendidos para outras áreas de conhecimento, como Física, Quimica, Biologia, etc.., onde existe uma série de conceitos abstratos como velocidade, equilibrio, pressão, etc., difíceis de serem compreendidos.

Nos Capítulos 2 e 3 faço uma análise do ensino de programação considerando duas vertentes básicas: o ensino convencional e o ensino auxiliado por sistemas computacionais especialmente desenvolvidos para este fim.

No **Capítulo 4** apresento a linguagem Logo tanto em seus aspectos computacionais como metodológicos. Neste capítulo procuro mostrar como Logo como linguagem diminui o salto entre algoritmos para pessoas e algoritmos computacionais e quanto aos aspectos metodológicos, a importância do respeito ao estilo, a importância do fazer, errar e depurar. Também são mostrados os casos em que existem falhas e é feita uma discussão da problemática em cada caso.

No **Capítulo 5** descrevo os aspectos teóricos advindos de pesquisas sobre modelos mentais. Estes aspectos embasaram todo o desenvolvimento deste trabalho.

No **Capitulo 6** descrevo sinteticamente o objetivo do trabalho e a metodologia utilizada no desenvolvimento da pesquisa.

No **Capítulo 7** descrevo o sistema implementado, mostrando os aspectos de apresentação do sistema e a estrutura de dados interna utilizada.

Nos **Capítulos 8, 9 e 10** apresento os resultados do uso do sistema em três situações diferentes.

Finalmente no **Capítulo 11** faço uma discussão dos resultados obtidos e apresento as conclusões.

Capítulo 2

Ensino Convencional de Programação

2.1 Introdução

O objetivo deste capitulo e do próximo, é retomar a discussão iniciada no capítulo anterior sobre ensino de programação, detalhando como isto vem sendo feito. A apresentação será subdividida considerando-se duas vertentes: o ensino convencional e o ensino auxiliado por sistemas computacionais, que será analisado no próximo capitulo.

Denomino de ensino convencional aquele que é feito estritamente com **"giz e quadro-negro"**, e portanto o termo convencional não tem implicações metodológicas mais profundas além da de espelhar o que mais corriqueiramente vem sendo feito com o ensino de programação. A abordagem que farei não será sobre aspectos teóricos de como deve ser feito o ensino nesta situação, e sim no que vem sendo efetivamente feito em sala de aula. Ao fazer esta abordagem estarei me baseando, fundamentalmente, no que é proposto em livros textos adotados em cursos introdutórios de programação [Koffman, 1985; Schneider, 1978; Keller, 1982] que definem a estrutura destes cursos. Além disso considerarei minha experiência efetiva como professora em diversos cursos introdutórios de programação, junto ao Departamento de Ciência da

Computação da UNICAMP.

Também discuto aspectos dos sistemas de depuração que são utilizados como auxílio no processo de programação. Atualmente estes sistemas de depuração estão presentes em todas os ambientes de programação e vem sofrendo constantes refinamentos. Esta discussão tem por objetivo apresentar o único recurso computacional de auxílio que os alunos de cursos introdutórios de programação podem fazer uso em cursos convencionais.

Dentro da categoria de ensino auxiliado por sistemas computacionais estarei analisando os tipos de sistemas que existem implementados no sentido de utilizar o computador como auxiliar no ensino de programação. Como poderá ser visto, as mais variadas metodologias de ensino estarão delineando tais sistemas.

2.2 Princípios

Tradicionalmente o ensino de programação é feito sob o paradigma procedural, utilizando linguagens imperativas ou procedurais como o Fortran, Pascal, etc.

Atendendo à este paradigma, o objetivo dos cursos é o de apresentar programação como uma atividade bastante complexa e que é subdividida em passos bem definidos que devem ser seguidos para obter um "bom" programa.

O princípio norteador dos cursos convencionais pode ser descrito pela seguinte afirmação, extraída de um dos livros texto amplamente utilizado:

"É uma idéia bastante errada a de que a melhor técnica para resolver um problema é sentar na frente do computador, começar a escrever um programa e continuar a escrever até que o programa esteja pronto, e ainda acreditar, que foi produzida uma solução válida. Não existe nada mais distante da verdade que esta idéia. Uma enorme quantidade de trabalho preparatório precisa ser feito antes de obter o código de uma solução em potencial. Esta preparação envolve passos tais como, definir exatamente o problema, clarificar quaisquer ambiguidades e dúvidas na proposta do problema, decidir como resolvê-lo, e descrever a solução em uma notação conveniente. Na realidade, se este trabalho preparatório for bem feito, a fase de codificação, a qual parece a mais importante para muitas pessoas, torna-se relativamente

simples e sem criatividade. Ela passa a ser simplesmente a tradução mecânica da solução do problema em comandos gramaticalmente corretos de uma particular linguagem de programação. O não entendimento deste princípio, é o **primeiro e maior erro** que se comete ao aprender programação." [Schneider, 1978, pg. 15]

De acordo com os princípios de programação estruturada [Dijkstra, 1976; Hoare, 1972] a tarefa de escrever um programa é subdividida em três fases: especificar, codificar e depurar.

Um programa é definido como uma sequência de passos que o computador deve seguir e o desenvolvimento de um programa deve iniciar com a descrição geral de seus passos básicos e gradualmente deve ser feito um refinamento dos passos até que tudo esteja descrito com "detalhes suficientes".

Introduz-se o conceito de algoritmo como um tipo de descrição da solução de um problema que explica o processo de solução, semelhante a uma receita.

Inicialmente são trabalhados alguns exemplos de algoritmos que geralmente explicitam sequências de instruções para execução humana, por exemplo, "como fazer um omelete de batatas". Introduzir a construção de algoritmos desta forma dá uma impressão errada do que venha a ser um algoritmo computacional, por dois motivos: primeiro, a linguagem utilizada geralmente é sem restrições e portanto ambigua; segundo, as instruções geralmente contém várias suposições que não precisam ser explicitadas, pois passam como senso comum ou bem entendidas, e mais tarde os estudantes se surpreendem com o nível de detalhe e as restrições de forma com que deve ser dada uma instrução, ao se programar computacionalmente uma tarefa.

Enfatiza-se que a construção do algoritmo, que corresponde a fase de especificação, é um passo essencial no processo de programação, pois conduz ao desenvolvimento de programas com o mínimo de erros. Preferencialmente um algoritmo é escrito em uma linguagem especial próxima, mas independente, da linguagem de programação e deve ser preciso, determinístico e finito. Para principiantes estas exigências não tem o menor significado.

Observando-se a forma como os alunos desenvolvem seus

programas durante estes cursos, de maneira alguma tem-se obediência a esta forma estabelecida de produzir um programa.

O que se observa é o aluno fazendo um desenvolvimento interativo do programa junto ao computador, mesmo utilizando linguagens herméticas como o Pascal, que em muito dificultam esta forma exploratória de desenvolver programas.

Se estas fases da atividade de desenvolver programas são tão enfatizadas durante os cursos introdutórios pergunta-se: Por que os alunos não assimilam a necessidade destes passos iniciais? Por que é tão difícil fazê-los compreender a necessidade da construção de um algoritmo?

Com isto não quero dizer que a abordagem esteja errada. Ela somente não é, na maioria das vezes, adequada em se tratando de principiantes, sem qualquer conhecimento do computador. Mais tarde, no decorrer de sua formação,os alunos realmente passam a adotar este padrão de desenvolvimento de programas. Isto ocorre com alunos de Computação e portanto futuros especialistas, quando do desenvolvimento de projetos sofisticados. Nesta fase do curso de bacharelado os alunos já estão absolutamente seguros de

como funcionam todas as construções da linguagem que estão trabalhando.

Outro aspecto a ser considerado é que com o advento dos microcomputadores o aluno passou a estar muito próximo à máquina. Quando os cursos de introdução a programação eram dados há 10 anos atrás, realmente o programa deveria ter o mínimo de erros possível, pois as possibilidades de "rodar" o programa eram bastante escassas. Hoje em dia isto não mais acontece. Inclusive os compiladores atualmente utilizados, por questões de otimização, não mais fazem um extenso relatório de todos os erros encontrados em um determinado programa. Os erros são apontados e corrigidos um a um. Portanto, mesmo o acerto da escrita do código é feito de forma bastante interativa. Por que a correção da semântica não pode ir pelo mesmo caminho? É claro, que corrigir um programa com erros semânticos é bem mais complicado e menos mecânico que corrigi-lo sintaticamente, mas mesmo assim pode ser auxiliado pelo computador, a partir das respostas que são dadas a cada alteração.

Portanto a abordagem dos cursos introdutórios, deve incentivar a utilização do computador o mais cedo possível, mostrando-o como

uma ferramenta interativa de solução de problemas. Isto em muito irá facilitar o aprendizado de programação e naturalmente este "pensar antes" irá ocorrer quando os alunos sentirem segurança e estiverem instrumentados para poder pensar. Não acredito que sejam criados os denominados **"vícios de programação"** que conduzam o aluno a sempre fazer programas no estilo "colcha de retalhos". O não entendimento de que é preciso conhecer a máquina que se está aprendendo a controlar e que o melhor modo é utilizando-a, errando e aprendendo com os erros, é um dos **maiores erros** que é cometido quando do desenvolvimento de um curso introdutório de programação.

Além disso, com esta abordagem rigorosa não se está considerando a grande e crescente gama de aplicações que não são suficientemente entendidas para permitir uma completa especificação. Nestes casos está se desenvolvendo a concepção de programação exploratória [Scheil, 1984], que torna-se a mais adequada graças a maior flexibilidade no projeto de programas. Dentro desta concepção tem-se o conceito de que desenvolver um programa é mais uma atividade de design [Simon, 1971; Schon, 1990; Park, 1990] que uma atividade de engenharia. Através de um desenvolvimento interativo, onde estão envolvidos projetista e

usuário, o programa final é conseguido através de um processo de aproximação, ou seja, múltiplas soluções intermediárias que sucessivamente ficam mais próximas da solução final.

Estas aplicações sob esta nova concepção de programação, estão sendo subsidiadas pelo desenvolvimento de ambientes e ferramentas de programação interativos [Sandewall, 1984; Teitelman, 1984; Teitelbaum, 1981; Wilander, 1984]. Grande facilidade também se tem obtido com o advento de estações de trabalho pessoais que proveem tanto um alto poder de computação, como também estilos de interação alternativos [Kernighan, 1984].

2.3 Apresentação dos Conceitos Básicos

A introdução da linguagem de programação é feita através de pequenos programas e geralmente programas com apenas comandos de escrita na tela do seguinte tipo [Keller, 1982]:

```
begin
 writeln('  *   ');
 writeln(' *  * ');
 writeln('******')
end
```

Novamente pode-se observar que não é considerada a dificuldade do aluno em compreender o funcionamento da máquina. Programas que são uma série de "writes"(comandos de saída) dão impressão errada, ou pouco auxiliam o entendimento, do funcionamento do computador. Isto porque não existe uma diferença muito grande entre o texto do programa e o resultado que é mostrado na tela. Fica confusa a diferença que deve ser ressaltada entre o texto e seu efeito quando executado.

Outro fato interessante que ocorre é quando da introdução do conceito de variável, um conceito muito importante em se tratando do paradigma procedural. Variável é definida como uma posição de memória a qual se dá um nome e um valor, e cujo valor pode ser alterado durante a execução do programa. Como é de senso comum entre programadores, ao se nomear variáveis devem ser utilizados nomes que de alguma forma lembrem a função da variável. Por

exemplo, se uma variável é utilizada para contar, pode ser chamada de **Cont**, se é utilizada para referenciar linhas de uma matriz, pode ser chamada de **Linha**, e assim por diante.

Isto é uma boa prática de programação, mas não é boa de se adotar junto com principiantes. Associando-se o nome da variável com sua função leva o iniciante a pensar que é necessário ter esta associação para que o "computador entenda o que deve ser feito". Pelo desconhecimento do aluno do real funcionamento da máquina ele é levado a acreditar que o computador possui uma capacidade de entendimento muito maior do que a real. Acredito que a prática de utilizar esta forma de nomeação deve ser reforçada, mas deve ser dada atenção a este fato que geralmente ocorre. Novamente, entender como a máquina funciona aparece como essencial.

Para facilitar o entendimento do conceito de variável, são desenvolvidos uma série de pequenos problemas, sendo prática efetuar a cada solução um "teste de mesa", ou seja, simular a execução do programa levando em conta apenas as alterações dos valores de variáveis [Keller, 1982; Schneider, 1978]. Geralmente os alunos acabam assimilando esta técnica e a adotam quando

efetuam depuração de seus programas. Inclusive a maioria dos modernos compiladores fornece esta facilidade de depuração.

Procedimentos, com parâmetros e escopo de validade de variáveis são atualmente introduzidos bem cedo nos cursos [Koffman, 1985; Keller, 1982]. Introduzir estes conceitos é essencial para que o desenvolvimento de programas seja feito como o desejado, ou seja, estruturadamente e de acordo com o paradigma procedural.

O conceito de procedimento é introduzido como uma forma de estruturar uma solução, facilitar a escrita e diminuir o tamanho do código final [Koffman, 1985]. O enfoque principal é na facilidade em resolver um determinado programa por partes [Keller, 1982].

Os problemas de entendimento que devem ser resolvidos são principalmente com relação ao controle de fluxo. Pois a partir da introdução de procedimentos, em linguagens como o Pascal por exemplo, não existe mais uma correspondência entre a forma estática com que são escritos os programas e a forma dinâmica de como são executados. Para facilitar este entendimento, como os

testes de mesa não são mais adequados, desenvolve-se outras formas de representação, como diagramas de execução [Kowaltowski, 1983], gráficos de rastreio [Collins, 1986], etc. Estas representações procuram espelhar como o programa será efetivamente executado retratando graficamente as mudanças de contexto quando da chamada e retorno de procedimentos. No decorrer deste capítulo faço uma análise de algumas das formas de representações desenvolvidas.

2.4 O Conceito de Recursão

Apesar do conceito de procedimentos ser introduzido bastante cedo no curso, procedimentos recursivos somente são tratados no final de um curso de um semestre [Keller, 1982]. Em cursos para alunos que não são do bacharelado de computação, muitas vezes não é abordado este conceito.

Geralmente o entendimento de recursão, enquanto um processo especial do computador não é muito salientado. A enfase é colocada na parte de solução de problemas, ou seja, identificar

problemas que podem ser resolvidos recursivamente [Schneider, 1978; Wirth, 1976]. Algumas afirmações extraídas de livros texto, ilustram este fato:

"Se há uma habilidade que distingue um programador Pascal novato de um experiente é a compreensão da recursão. Um subprograma é recursivo se é definido em termos de si mesmo. Não é difícil inserir dentro de um subprograma uma chamada a si mesmo. O que é difícil é reconhecer situações em que são apropriadas as chamadas recursivas" [Collins, 1986, pg. 87].

"Um algoritmo é denominado recursivo se ele é definido em termos de si mesmo. Em um algoritmo iterativo uma parte da solução pode ser repetidamente executada, mas um algoritmo recursivo chama para reexecução o algoritmo inteiro, desde o começo. A idéia fundamental por trás de uma solução recursiva é definir um problema em termos de uma versão similar de si mesmo"[Schneider, 1978, pg. 37].

Dificilmente o conceito de recursão é entendido sem que se explicite de alguma forma como o computador efetivamente processa um código recursivo. Isto pode ser observado no fato de que,

mesmo em problemas cuja a definição é recursiva (fatorial, exponencial, etc...), os alunos não apresentam a solução recursiva. Soluções recursivas são apresentadas somente para o caso de problemas cuja solução não recursiva é muito difícil e complexa, como é o caso do clássico problema da Torre de Hanói.

Mas a apresentação de uma solução clássica como esta da Torre de Hanói, não implica que o conceito de recursão esteja entendido. Para exemplificar, ao final de um curso de um semestre de programação, solicitei que a classe fizesse o procedimento que resolvia o problema da Torre de Hanói; 80% dos alunos apresentou a solução recursiva correta e desses 80% apenas 20% conseguiu descrever como o procedimento funcionava. Portanto pode-se supor que, usar recursão de forma sofisticada não implica em seu entendimento.

Para verificar se minha hipótese de que os alunos estavam usando recursão sem entender, tinha possibilidades de estar correta, fiz um pequeno teste junto à uma classe de alunos do curso de bacharelado em Ciência da Computação da UNICAMP. Os alunos que reponderam estavam terminando o terceiro semestre do curso, o que significa que já tinham tomado todos os cursos que

ensinam programação Pascal, dentre os de seu currículo.

O teste foi baseado no trabalho de Kahney descrito em [Kahney, 1989]. Foi apresentado aos alunos o seguinte problema:

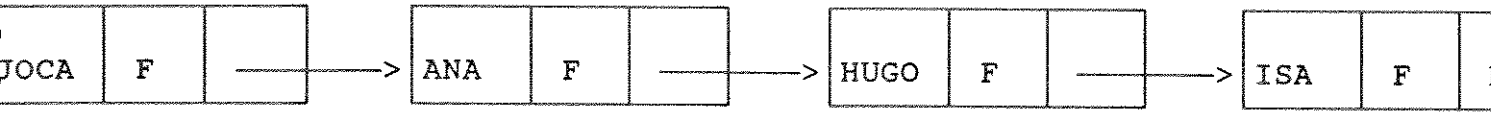
Deseja-se escrever um programa que faça a seguinte inferência:

"Se uma pessoa X tem GRIPE, então uma pessoa Y que beija X também tem GRIPE, e então a infecção se propaga para a pessoa que Y beija, e assim por diante"

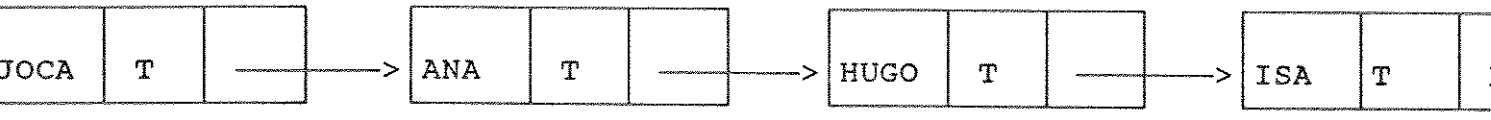
Dada a seguinte estrutura:

```
Type  p = ^reg;

      reg = record
            nome:string;
            gripe:boolean;
            beijo:p
            end;
```



Deseja-se fazer um procedimento que a altere para:



Serão apresentadas três soluções nomeadas de SOLU1, SOLU2 e SOLU3

Gostaria que voce considerasse cada solução por vez e respondesse:

a) se o procedimento faz ou não faz o desejado

b) se o procedimento faz, diga como ele faz (em suas próprias palavras) ou se ele não faz diga porque ele não faz (novamente com suas próprias palavras).

```
procedure SOLU1 (ap:p);
begin
  if ap^.beijo <> nil then ap^.beijo^.gripe := true
end;{SOLU1}


procedure SOLU2 ( ap:p);
begin
ap^.gripe:=true;
if ap^.beijo <> nil then SOLU2 (ap^.beijo)
end; {SOLU2}


procedure SOLU3 ( ap:p);
begin
if ap^.beijo <> nil then SOLU3 (ap^.beijo);
ap^.gripe:= true
end; {SOLU3}
```

Considero um problema simples para alunos que faziam procedimentos recursivos bastante complexos tais como, avaliação

de expressões, percursos de árvores nas mais diferentes formas, etc., e os faziam com bastante facilidade.

No total 37 alunos responderam ao problema e desses, 12 responderam incorretamente, portanto cerca de 30% respondeu incorretamente. A fonte principal de erro foi quanto a análise do procedimento SOLU3, que é o que justamente verifica a compreensão de recursão no sentido mais amplo, ou seja, com operações a serem efetuadas no retorno.

Algumas respostas selecionadas exemplificam o padrão de respostas incorretas obtido:

"O procedimento SOLU2 funciona, pois deixa a pessoa corrente gripada e repete as operações para a pessoa seguinte, se houver. O procedimento SOLU3 não funciona, deixa apenas a última pessoa gripada e as outras do jeito que estavam"

"O procedimento SOLU2 está correto, pois ao ser feita a chamada recursiva, é feita, imediatamente, a atribuição de valor, antes de uma nova chamada. O procedimento SOLU3 não está correto,

pois haverá sucessivas chamadas recursivas sem que seja feita a atribuição"

"O procedimento SOLU2 faz exatamente o desejado, a cada "loop" os respectivos nós recebem true. SOLU3 apenas percorre a lista ligada não alterando os campos booleanos dos nós, sendo que estes permanecem com o valor "false"; o único nó a receber "true" é o último"

Através destas respostas verifica-se que estes alunos tem como modelo de recursão o mesmo modelo de iteração, denominado modelo de "loop" [Kahney, 1989]. Esta é uma forma que as pessoas tem para entender recursão e que sempre funciona quando se tem a denominada "recursão de cauda", que independe de se considerar retorno, que sem dúvida é mais ligado ao entendimento de como a máquina executa procedimentos recursivos. Como tais alunos conseguem escrever procedimentos recursivos complexos? Minha hipótese é de que eles adquirem padrões de procedimentos recursivos que são ensinados e cobrados como avaliação, mas cujo funcionamento para eles é um mistério.

O entendimento de recursão acontece quando a pessoa adquire o modelo de cópia [Kahney, 1989], ou seja, quando consegue ter um modelo mental de que a cada chamada uma cópia do procedimento é ativada com diferentes parâmetros, e que os retornos são feitos cópia a cópia, na ordem inversa de sua criação.

Para ajudar a visualizar este processo dinâmico, muito distante da forma estática do código do programa, os cursos convencionais fazem uso de representações gráficas que procuram espelhar o funcionamento dinâmico do programa. São representações com características bem diversas, e foi com base nestas representações que defini a apresentação do sistema que implementei. A seguir discuto algumas destas representações.

2.5 Representações Computacionais Estáticas

Tradicionalmente são utilizadas representações da execução de um programa. Elas são feitas com o objetivo de ajudar a entender aspectos de programação que estão muito ligados ao funcionamento do programa quando em execução, ou seja, sequencialidade de execução de comandos, controle de fluxo, passagem de parâmetros, escopo de validade de variáveis (para o caso de escopo dinâmico), recursão, etc.

Vou fazer um apanhado de tais representações e dos aspectos que cada uma representa. Tratarei especificamente de algumas que se adequam a representar programas estruturados na forma de procedimentos.

2.5.1 Diagrama de Execução [Kowaltovski, 1983]

Nesta representação, que é a mais utilizada em cursos introdutórios de computação na UNICAMP, a execução de um programa é apresentada sob a forma de retângulos encaixados denominados

registros de ativação. Toda vez que é chamado um procedimento, é aberto um novo retângulo que será fechado ao término da execução do respectivo procedimento. Este retângulo aparecerá graficamente encaixado no retângulo do procedimento chamante. No topo de cada retângulo são listados os parâmetros e as variáveis locais àquele registro de ativação, juntamente com seus respectivos valores. Para identificar as variáveis globais existe um retângulo mais externo que representa o nível principal de execução.

Vamos exemplificar apresentando o diagrama de execução do procedimento CONTAR, escrito em Logo:

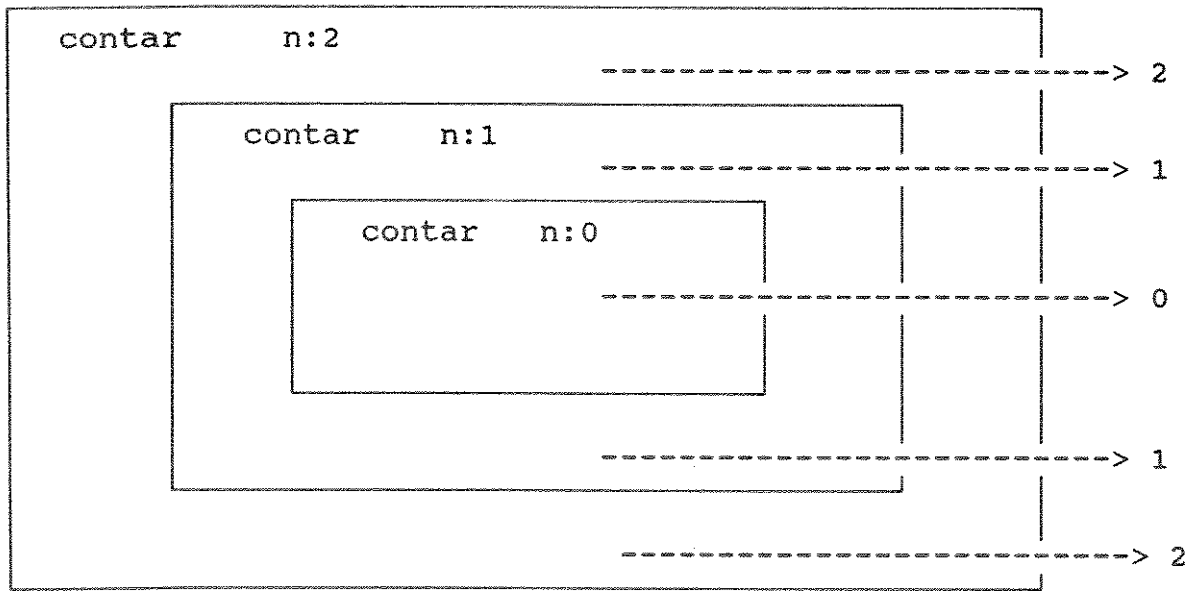
```
aprenda contar :n
escreva :n
se :n = 0 [ pare ]
contar :n-1
escreva :n
fim
```

Este procedimento escreve os valores n, n-1, n-2,..., 0, 1, 2, ... n

Efetuando a chamada:

? contar 2

tem-se a representação da execução no seguinte diagrama:



Pode-se observar que, de forma clara, é delineado o escopo correspondente de cada "encarnação" do parâmetro n . Fica também claro o retorno ao escopo anterior (valor anterior) ao terminar a execução de uma chamada.

As flechas que saem dos retângulos indicam as saídas que são produzidas e, a grosso modo, em qual momento da execução elas ocorrem.

Perde-se nesta representação a visão do fluxo de execução mais primário, ou seja, de instrução por instrução. Por exemplo, não se consegue ter claro nesta representação a execução de um comando repetitivo. Em se tratando de linguagens como Logo onde pode-se criar variáveis globais em qualquer ponto de um procedimento e sem ser preciso declarações prévias, o esquema de representação teria que sofrer alteração de modo a poder representar o dinamismo da área de trabalho.

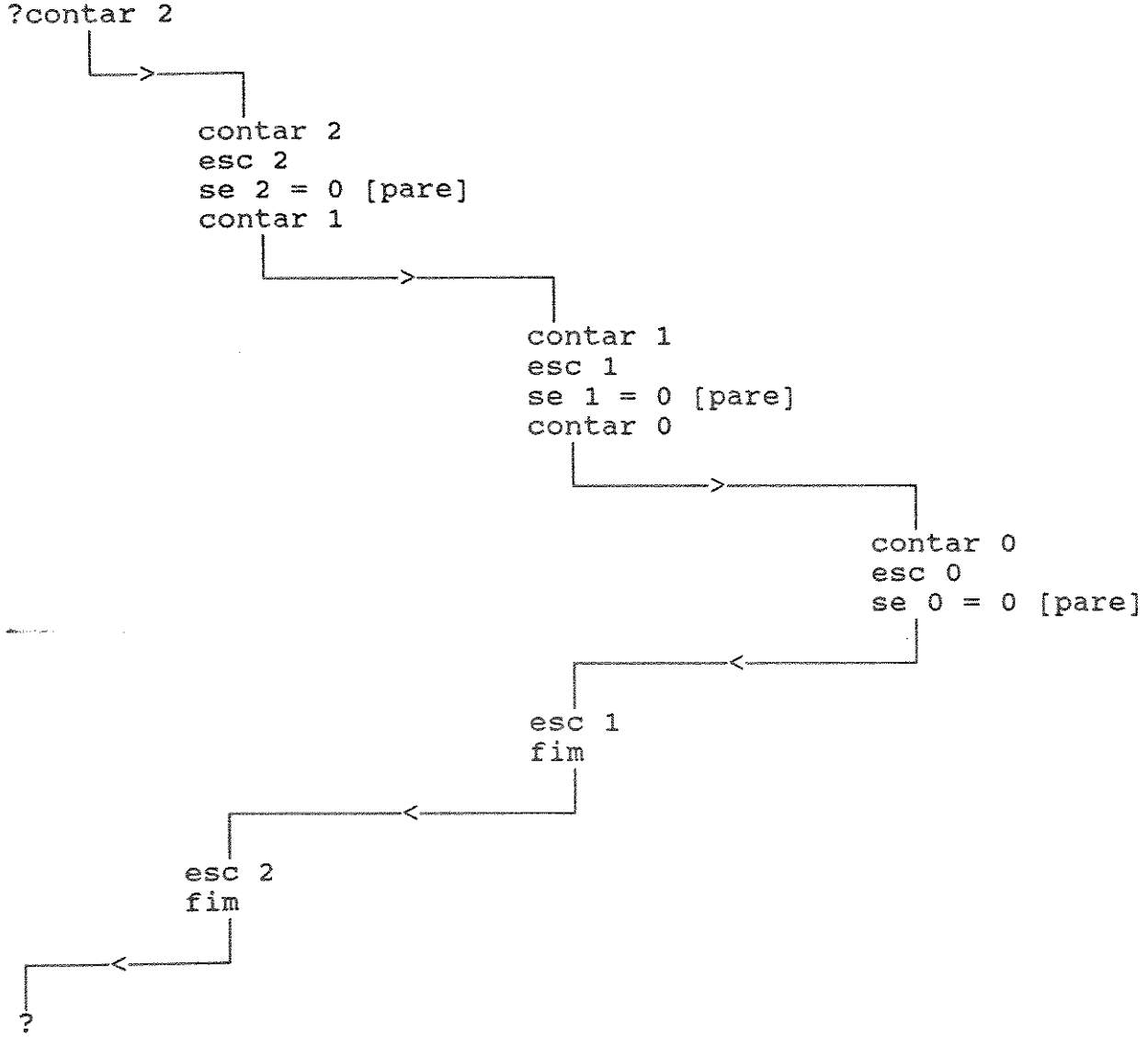
2.5.2 Cadeia de Execução [Valente, 1988b]

Nesta representação, que é bastante utilizada em cursos introdutórios da linguagem Logo, é feita a expansão do código do procedimento, substituindo-se a cada chamada o respectivo valor das variáveis locais e parâmetros.

Para exemplificar esta forma de representação vamos utilizar o mesmo procedimento CONTAR usado na representação anterior.

```
aprenda contar :n
esc :n
se :n = 0 [ pare ]
contar :n-1
esc :n
fim
```

Para a chamada **contar 2** teríamos:



O objetivo é ressaltar a imagem de que a chamada de um procedimento faz com que uma cópia do mesmo seja ativada, com novos valores de variáveis. Além disso fica claro em qual instrução é interrompida a execução de um procedimento ao se efetuar uma chamada e onde deverá ser reiniciada a execução no retorno, ressaltando aspectos de controle de fluxo. Fica difícil visualizar o processo de retorno de valores, quando da definição de expressões, tendo-se a impressão que um mesmo comando é executado duas vezes. Perde-se nesta representação o contexto de validade das variáveis, principalmente quando se tem regras de escopo estáticas e não dinâmicas como no Logo.

A principal vantagem desta representação em relação ao diagrama de execução é a clareza com que é mostrada passo-a-passo a execução de um determinado procedimento.

2.5.3 Diagrama da Sequência de Chamadas [Collins, 1986]

Nesta representação aparecem combinadas a representação de escopo de variáveis e sequência de execução de instruções. Como

poderá ser observado, fica visualmente difícil perceber a ordem de chamada dos procedimentos e o momento de retorno (controle de fluxo) , pois isto é denotado por flechas e não por encaixamentos. Tanto isto é difícil que especifica-se em cima de cada registro de ativação a qual saída corresponde.

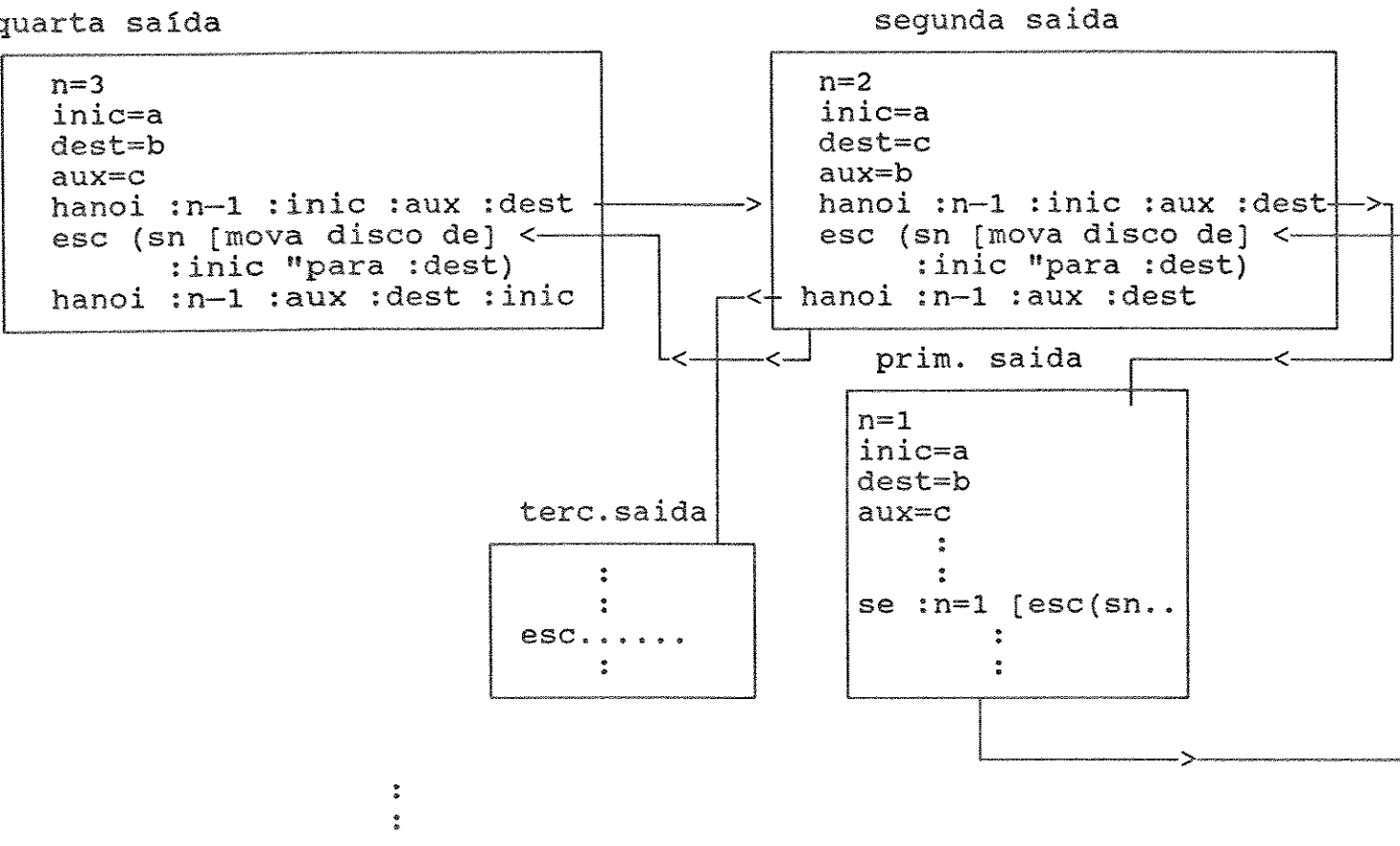
Para exemplificar vamos utilizar o procedimento que resolve o problema da TORRE DE HANOI. O diagrama não foi feito completamente, apenas é mostrado um trecho para demonstrar a forma de representação.

```
aprenda hanoi :n :inic :dest :aux
se :n=1 [esc (sn [mova disco 1 de] :inic "para :dest ) pare]
hanoi :n-1 :inic :aux :dest
esc (sn [ mova disco] :n "de :inic "para :dest)
hanoi :n-1 :aux :dest :inic
fim
```

considerando-se a chamada

? hanoi 3 "a "b "c

teríamos a representação a seguir:



O que deve ser ressaltado quanto a esta representação é a dificuldade em conseguir através de uma única representação, mostrar diferentes aspectos da execução. Isto resultou em uma

representação bastante difícil de ser acompanhada e entendida, apesar dos aspectos mostrados serem importantes e complementares.

2.5.4 Tabela de Rastreio ("Trace Table") [Carvalho, 1982]

Esta é uma forma de fazer **teste de mesa estruturado.**

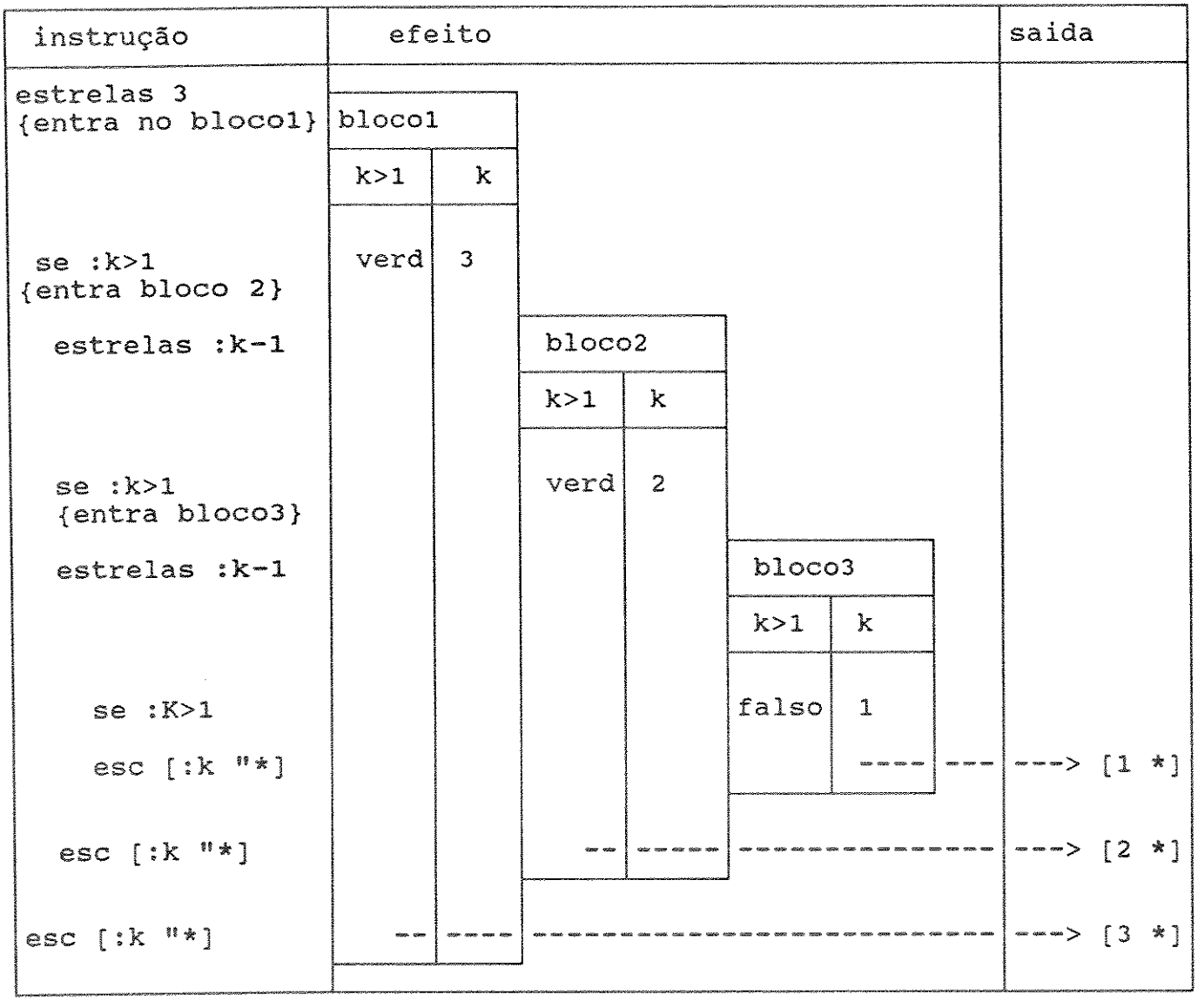
Nesta representação para mostrar o efeito de chamadas de procedimentos, são estabelecidos cabeçalhos para um conjunto de colunas que correspondem à todas as variáveis locais (as declaradas e os parâmetros), toda vez que ocorre a chamada de um procedimento. Também são colocadas colunas específicas para as instruções de controle de fluxo que estão sendo executadas. Define-se uma coluna para as saídas na ordem em que ocorrem.

Os blocos são numerados de acordo com o nível da chamada.

Para exemplificar vamos utilizar um procedimento bastante simples, recursivo, que imprime uma sequência de listas da forma [n *], com n variando de 1 a k.

```
aprenda estrelas :k
se :k>1 [estrelas :k-1]
esc [:k "*]
fim
```

Para a chamada **? estrelas 3** teríamos:



Novamente pode-se observar a combinação de aspectos do processo de execução, mostrados em uma única representação, complicando por demais o entendimento. Nesta representação, como complicador ao entendimento, é feita uma representação da execução a nível de instrução como o caso do comando condicional **se**.

Existem outras representações definidas na literatura, mas as apresentadas exemplificam bem as características gerais. Todas as formas são muito boas como auxílio ao entendimento, o que somente comprova a necessidade de se ter um modelo do processo de execução. Excetuando-se as características negativas, já levantadas, de cada representação, existe um sério problema em se utilizar tais representações. Construí-las é um processo bastante tedioso, e geralmente é feito somente uma vez, para um certo conjunto de valores, quando o interessante para absorver realmente o processo, seria construí-las para diferentes valores e das diferenças abstrair o comportamento. A importância desta exploração pode ser comprovada quando do uso do sistema que implementei, onde facilmente o usuário podia reexecutar o procedimento, com diferentes valores, e geralmente eles o faziam muitas vezes para o mesmo procedimento.

2.6 Ambientes de Depuração

Atualmente, os ambientes de programação desenvolvidos para microcomputadores, possuem um sistema de auxílio à depuração de programas. Estas ferramentas estão sendo muito incrementadas. Isto denota uma preocupação no sentido de utilizar o computador como auxílio no processo de programação.

Os primeiros depuradores, somente permitiam acesso à uma infindável lista de variáveis e seus valores correspondentes. Muitas vezes o enderêço da variável também era listado. Para principiantes, que não estão seguros do funcionamento da linguagem, esta tabela de valores não era muito útil.

Atualmente os depuradores, continuam fornecendo o valor das variáveis e seus enderêços, mas adicionalmente permitem inspecionar a pilha de execução, alterar o valor de variáveis durante o processo de depuração, executar o programa passo a passo, etc.

Para exemplificar, vou descrever suscintamente as características básicas do depurador implementado no sistema

Turbo Pascal, versão 5.5. Neste sistema o processo de depuração é acionado através de uma tecla de função, aparecendo na tela o texto do programa e em vídeo reverso o primeiro comando executável do programa. O conceito de comando executável é o de comandos que geram código. Dessa forma a palavra reservada **Begin**, que indica o início de um comando composto é indicada como comando executável pelo depurador.

Depois de ativado o processo de depuração, pode-se então executar o programa linha a linha. As variáveis para as quais se quer verificar os valores sucessivos durante o processo de execução devem ser previamente selecionadas. Os valores e os nomes das variáveis são mostrados em uma janela especial denominada "Watch" que sempre está presente na tela padrão.Caso nenhuma variável ou expressão seja selecionada, nenhum valor será mostrado e a respectiva janela pode ser eliminada pelo usuário.

Quando se tem a chamada de procedimentos, pode-se escolher o modo como esta chamada vai ser rastreada: de uma só vez, como a execução de um comando qualquer, ou entrando no procedimento e executando-o linha a linha. Caso se deseje investigar a pilha de execução ao longo de uma sequência de chamadas de procedimentos,

deve-se acionar uma outra tecla de função e aparecerá na tela uma janela com as sucessivas chamadas ainda ativas e os respectivos valores dos parâmetros. Esta janela não permanece na tela depois de acionada, ou seja, ao continuar a execução ela desaparece.

O programa não precisa ser seguido a partir do primeiro comando, podendo-se selecionar a partir de qual comando deve iniciar o processo de depuração. Isto é feito com o objetivo de auxiliar a depuração no ponto onde acredita-se que esteja o erro. Também pode-se adicionar ao texto do programa pontos de depuração, denotados "breakpoints". Com isto, pode-se efetuar uma outra modalidade de depuração, não mais passo a passo, parando a execução somente nos pontos marcados.

Pode-se também ter acesso interativo ao valor de variáveis e expressões, acionando uma outra tecla de função. Este acesso interativo permite que se altere o valor corrente de variáveis e a execução prossegue com estes novos valores.

Estas são as principais características do depurador do sistema Turbo Pascal, que com poucas diferenças são similares às de outros depuradores de ambientes de programação de outras

linguagens imperativas. É sem dúvida uma poderosa ferramenta de auxílio à programação. Sem dúvida também, não é dirigida à principiantes. O objetivo é auxiliar a depuração de programas extensos e muito complexos.

Pode-se observar que o mesmo paradigma que governa o uso da linguagem Pascal, também guia o uso do depurador. Tudo o que se deseja ver e observar deve ser definido "a priori". O uso é extremamente imperativo, ou seja, todas as ações especiais do depurador são acionadas sob comando.

Principiantes e não especialistas, dificilmente fazem uso desta ferramenta. Quando não existia este tipo de depurador, era extremamente difícil fazer com que os alunos adquirissem um método de auxílio a depuração. O que se tentava mostrar é que inserindo diversos comandos de saída, que escrevem valores de variáveis significativas ou então que escrevem mensagens indicando pontos significativos da execução, em muito facilita a depuração. Qual era a dificuldade em passar esta idéia, a princípio tão simples? Os alunos não conseguiam identificar variáveis e pontos de execução significativos. Principiantes, não

conhecem o funcionamento da linguagem e portanto tudo a princípio é significativo. Mas se tudo é selecionado, a confusão permanece.

A dificuldade no uso do depurador computacional é a mesma. É preciso saber programar, saber como tudo funciona, para poder selecionar os aspectos significativos a serem observados. E este uso do computador não fornece o auxílio necessário para adquirir este conhecimento.

2.7 Conclusão

Pode-se perguntar: O ensino convencional é eficaz?

Sem dúvida, sim. Inúmeras pessoas aprendem a programar nestes cursos. Mas esta análise que foi feita, tem como preocupação básica a crescente dificuldade em motivar os alunos, de forma a que se interessem por programação. E como já afirmei na introdução, isto é um paradoxo com a realidade, onde um interesse por programação é crescente. Não é concebível, que a cada ano, mesmo em cursos exclusivos de alunos de bacharelado em Computação, o nível de reprovação e abandono do curso cresça

consideravelmente.

Nos últimos dez anos, o ensino de programação vem se mantido sob os mesmos princípios e com a mesma abordagem. Isto também é um paradoxo, em se considerando o avanço da área de computação nestes 10 anos. Evoluiu-se de máquinas do tipo IBM1130 para estações pessoais de trabalho. A natureza das aplicações modificou-se radicalmente.

É preciso repensar nos princípios e metodologias de ensino, bem como em linguagens a serem utilizadas e em ferramentas e ambientes computacionais que possam ser desenvolvidos com o intuíto de auxiliar o aprendizado.

Decorrente desta análise, implementei um sistema computacional que pretende auxiliar o ensino de conceitos básicos, cujo entendimento dependa de ter um modelo de como o programa é efetivamente executado.

No sistema que implementei, como poderá ser visto em detalhes no Capítulo 7, foram definidas representações gráficas da execução, baseadas em características das representações

aqui descritas. Evitou-se definir uma única representação que mostrasse os aspectos da execução que se queria ressaltar, pois como foi visto, isto deixa a representação por demais complexa e de difícil entendimento.

Utilizou-se o recurso de dividir uma tela em diversas janelas, onde em cada uma é mostrado, de forma bastante simples, um aspecto da execução e a simultaneidade da apresentação é que garante a observação da complementariedade dos aspectos.

Como poderá ser observado na descrição do sistema, optou-se por basear a representação nas características da representação de diagrama de execução, representando a sequência de chamadas por retângulos encaixados, e as idéias de cadeia de execução, onde são ressaltados os aspectos de controle de fluxo, comando a comando. Tudo isto aliado a telas de entrada e saída, e de "display" do valor das variáveis.

A característica mais importante que foi adicionada foi a dinâmica da representação. Como o objetivo é o de auxiliar o entendimento, acredita-se que em se tratando de um processo dinâmico muito entendimento já deve ter sido assimilado para

chegar a compreender uma representação estática. Além disso, a interação que é permitida durante a execução, viabiliza toda uma fase de experimentação que advoga-se como essencial no entendimento de qualquer processo.

Capítulo 3

Sistemas Computacionais no Ensino de Programação

3.1 Introdução

O objetivo deste capítulo é analisar o que vem sendo feito, com relação à utilização de sistemas computacionais, no ensino de programação.

Pode-se subdividir os sistemas computacionais existentes em três categorias bastante amplas, de acordo com os objetivos educacionais que propõem e os tipos de conhecimento que empregam.

1. TUTORES: são sistemas que atuam no sentido de tomar decisões sobre como, e em que ordem conceitos de programação devem ser introduzidos e acompanhar o progresso do estudante no domínio destes conceitos. Uma variação destes sistemas são os denominados INSTRUTORES ("Coaches"). Eles monitoram a fase de solução de problemas e portanto enfocam uma parte mais restrita do processo de programação. Estes sistemas acompanham o modo como o estudante resolve um problema, fazendo comentários quando alguma coisa parece errada, dando ajuda quando solicitado ou então levando o aluno a resolver o problema de um modo pré-especificado.

2. DEPURADORES ("BUG FINDERS"): são sistemas que detetam o que há de errado com o programa do estudante e então dão sugestões sobre o que deve ser feito. Estes sistemas diferem dos tutores e instrutores, no sentido que eles esperam o programa completo antes de efetuar qualquer análise. A deteção é feita para erros lógicos procurando com isto identificar falhas conceituais na escrita de programas.

3. AMBIENTES DE SUPORTE: é a categoria mais difusa de sistemas e consiste basicamente daqueles sistemas que proveem um meio de tornar a interação do estudante e o sistema de programação mais efetiva e mais compreensível. Ambientes de suporte englobam ferramentas tais como depuradores inteligentes, programas de rastreio (trace), editores, linguagens de programação, etc. De modo geral, são toda espécie de ferramenta computacional construída objetivando aprendizes e não especialistas de programação.

As categorias relacionadas não são rígidas, pois em muitas delas encontra-se sistemas que apresentam características comuns. Por exemplo, um sistema de qualquer categoria tem que efetuar uma deteção "inteligente" de erros e possuir ferramentas bem

planejadas de suporte à escrita de programas.

A seguir comentarei de maneira mais extensa cada uma das três categorias.

3.2 TUTORES / INSTRUTORES

A estrutura clássica para sistemas de ensino inteligentes descritas por Boulay em [Boulay,1987] foi fixada por Hartley em [Hartley, 1973]. Para ele, tais sistemas são construídos em torno de quatro fontes de conhecimento:

1. Conhecimento ou habilidade que é ensinada.

Dá ao sistema a capacidade de resolver os problemas que são propostos aos estudantes para poder tanto julgar e comentar as respostas como responder perguntas propostas pelos estudantes.

2. Conjunto de ações de ensino ou táticas de ensino.

Estas ações compreendem fazer um comentário positivo ou

negativo, prover exemplos, solicitar explanações do aluno sobre um determinado assunto, propor problemas, propor ao estudante encontrar um contra-exemplo, etc. Propor problemas geralmente envolve a existência de um gerador de problemas que tanto pode selecionar um problema de uma base de problemas como também dinamicamente prover parâmetros para um esquema geral de problemas.

3. Modelo do estado corrente do estudante ou diagnóstico cognitivo do estudante

Compreende o entendimento atual do sistema a respeito da história, capacidades, conhecimento, objetivos e motivação do estudante. Este tipo de conhecimento é o que mais vem sendo aperfeiçoado nos sistemas.

4. Conjunto de regras de meio-fim ("means-end") ou estratégias de ensino.

Determina quais ações de ensino a serem adotadas considerando os objetivos do sistema e o corrente modelo do estudante. Estas

regras guiam o comportamento do sistema e definem o modelo do tutorial.

Diagnóstico cognitivo, estratégias de ensino e conhecimento a ser ensinado são as entradas de um sistema tutorial, ou seja, a informação que forma a base das decisões do tutorial. As táticas de ensino representam sua saída, seu comportamento perante o usuário.

Em se tratando de tutores para ensino de programação, os sistemas devem possuir conhecimento de programação. Geralmente este conhecimento é subdividido em:

1. conhecimento sobre a sintaxe e semântica da linguagem de programação utilizada.

Conhecimento sintático envolve todo conhecimento sobre a forma da linguagem de programação, ou seja, como um código deve ser escrito. Observa-se que principiantes erroneamente e com frequência vem este aspecto como central no aprendizado de programação e o uso de sistemas tutores podem vir a agravar esta visão errônea se não possuirem flexibilidade de lidar com códigos

que não estejam absolutamente corretos, relevando erros sem significado conceitual.

Conhecimento semântico pode ser subdividido em aspectos procedurais e declarativos. Como procedurais se entende o conhecimento sobre como a máquina conceitual descrita pela linguagem de programação efetivamente trabalha. Este conhecimento é que possibilita o sistema ou o estudante predizer qual comportamento um código irá ter e o inverso disto, ou seja, observado um dado comportamento retornar o código. Os aspectos declarativos são os diretamante relacionadas à estrutura da linguagem, ou seja, o conhecimento sobre o que faz um determinado comando, as propriedades das estruturas de dados e seus tipos, etc.

2. conhecimento tático e estratégico de como as estruturas da linguagem se combinam e como decompor um problema nestas estruturas.

Conhecimento sobre a decomposição de problemas é amplamente tático, envolvendo conhecimento sobre como decompor um problema em sub-problemas que podem ser resolvidos mais facilmente. Saber

que isto é possível ser feito é lugar comum, mas saber como fazer para uma dada classe de problemas não é tão óbvio. E o fator complicador é definir o quanto uma determinada maneira de subdividir um problema está de acordo com o estilo do estudante ao qual o problema foi proposto. Esta última preocupação não está presente em nenhum sistema implementado até o momento onde observa-se o uso de estratégias padrão de produção de software tais como: conhecer quão grande um programa pode ser para resolver um dado problema, número máximo de comandos por procedimento, etc.

Também é importante o conhecimento sobre trechos padrão de código que reconhecidamente produzem certa classe de efeitos e o modo de interligá-los. Se não houver grande flexibilidade neste aspecto pode ser que o estudante assimile "padrões que funcionam" sem contudo efetivamente entender como funcionam.

3. conhecimento pragmático de como conduzir um diálogo efetivo com o ambiente de programação.

Definir qual metodologia a ser adotada no desenvolvimento de um programa, saber como testar e depurar um programa, saber que

atitude tomar quando em situação de impasse, saber como editar, compilar e interpretar as mensagens do compilador (interpretador), etc..

Esta é a gama de conhecimento sobre programação desejável, mas não está presente em sua totalidade nos sistemas atualmente implementados.

Ao se adotar um tutor computacional está sem dúvida sendo feita uma opção de ensino individualizado. Com isso, acredito que a grande promessa dos tutores computacionais está na perspectiva de se ter sistemas adaptáveis a cada estudante que o utiliza. Efetuar o atendimento a todos os estilos cognitivos implica em ter uma perspectiva longitudinal, ou seja, focalizar as diferentes necessidades cognitivas de um único aprendiz a cada tempo, sem ter a preocupação com diferenças entre categorias de aprendizes que são por sua natureza mais estáveis e amplas.

Para que isto ocorra muitas das características que qualificam um sistema como tutor devem ser especialmente implementadas.

Com relação ao conhecimento a ser ensinado o tutor precisará

fazer uma distinção entre o conhecimento que está sendo ensinado e o formato no qual é apresentado, e deverá ser capaz de gerar diferentes apresentações, escolhendo a forma mais benéfica para o aprendiz no momento.Isto se deve ao fato do aprendiz poder adquirir diversas representações internas do que está sendo ensinado, todas igualmente válidas do ponto de vista educacional.

O computador, a princípio, poderá ser programado para gerar exatamente aquela questão, explicação, exemplo, contra-exemplo, problema prático, ilustração ou demonstração a qual seja de melhor ajuda ao estudante. Esta ambiguidade é uma séria dificuldade, pois qualquer implementação precisa adotar uma representação do conhecimento a ser ensinado e então, ou se tem uma grande redundância de informação com formatos diferentes, ou então, busca-se uma representação "canônica" que possibilite gerar dinamicamente diferentes representações. Esta última alternativa vem sendo fonte de pesquisa dos grupos envolvidos no desenvolvimento de tutores.

Muito esforço vem sendo feito no sentido de aperfeiçoar o denominado modelo do estudante e com isso vem sendo ressaltado o fato de que planejar e projetar instrução a mais útil possível

para o estudante não é tarefa simples, mesmo com a ajuda de uma poderosa descrição do estudante.

Os diferentes modelos de diagnóstico cognitivo existentes formam uma progressão em direção a uma mais poderosa descrição do estudante.

Existe o modelo de sobreposição que pressupõe o conhecimento do estudante como um subconjunto do conhecimento de um perito e que aprendizagem é o processo de adquirir progressivamente um subconjunto mais completo das unidades do conhecimento do perito. Mas como nem toda forma de conhecimento pode ser hierarquizada, pode-se ter estudantes que possuam o mesmo conhecimento final conhecendo coisas bastante diferentes. Um sistema que emprega tal modelo, entende seu estudante como uma espécie de perito com conhecimento pobre e não leva em conta o fato de que o estudante pode ter um diferente, ao invés de mais simples, conhecimento quando comparado a um perito.

Existe também o diagnóstico feito através do erro. A evidência empírica que o fundamenta é que aprendizes não somente falham ao adquirir conteúdo apresentado durante instrução, como também os

representam errado.

Este tipo de diagnóstico usualmente trabalha sobre dados de performance, isto é, o padrão de respostas corretas e incorretas dentro de um conjunto de problemas. O procedimento inferencial o qual é usado neste tipo de diagnóstico é baseado numa biblioteca de erros geralmente construída empiricamente. Dado um estudante em particular, é possível computar qual o erro ou combinação de erros melhor justifica as respostas incorretas deste estudante. As dificuldades básicas de construir estes sistemas são:

a) estabelecer a biblioteca de erros

b) inventar métodos, os quais computem a combinação de erros que melhor se encaixa e que sejam suficientemente eficientes para fornecer um diagnóstico em tempo razoável.

A promessa pedagógica da descrição por erro é de possibilitar uma instrução remediadora. Tendo conhecimento sobre que erro em particular o estudante está cometendo, tem-se possibilidade de ajudá-lo a superá-lo. Entretanto decidir qual o melhor remédio para um particular erro não é tarefa trivial. Por isso sistemas

que adotam este tipo de descrição não são comuns e os existentes não possuem uma estratégia remediadora bem definida, estando ainda na fase de definir qual a melhor forma de reportar o erro e certamente, relatar o erro não é a única forma de ensino remediador.

Este tipo de diagnóstigo é utilizado em alguns depuradores inteligentes, como será visto na próxima parte deste capítulo.

Todos estes modelos são ditos declarativos, existindo a proposta mais prometedora dos modelos simuláveis ou procedurais, A princípio é uma poderosa descrição permitindo executar a representação do estudante sempre que necessário e talvez fazer predições detalhadas sobre sua performance. Ainda não existe nenhuma implementação que funcione com esta idéia de diagnóstico. Algumas tentativas semelhantes estão implementadas no sistema Greaterp [Anderson, 1984; Anderson,1985]

Deve ser levado em conta que ensino inteligente não consiste de sequências de ações não relacionadas. O esforço do tutor é estruturado; ele coordena ações individuais de ensino sob um plano de como transmitir conhecimento relevante. O propósito da

componente de diagnóstico cognitivo de um sistema tutor inteligente deve ser a de dar suporte a este plano instrucional.

Quanto as táticas de ensino, se um tutor tem um número limitado de ações para escolher ele não pode adaptar seu ensino as necessidades cognitivas do estudante. Enquanto um estudante precisa de uma definição, outro precisa de uma explanação e um terceiro pode aprender melhor a partir de um problema prático. De modo a prover instrução adaptativa, o tutor precisa ter uma grande variedade de escolha de táticas de ensino.

Por outro lado, o conjunto de táticas de ensino em um sistema é condicionado à estratégia de ensino adotada; a menos que a estratégia possa identificar circunstâncias sob as quais uma tática em particular possa ser evocada, a tática em si não aumenta o poder do sistema.

Da mesma forma que o diagnóstico, o repertório comportamental do sistema é logicamente secundário à estratégia de ensino e este é o grande gargalo dos sistemas atualmente implementados.

Estratégia de ensino nada mais é que a descrição de como ensinar, como gerar uma sequência de táticas de ensino que possam com sucesso transmitir o conhecimento a um estudante. Mas mesmo tendo-se milhões de professores dispensando milhões de horas todo ano ensinando milhões de estudantes, ninguém parece saber como ensinar. Não existe nenhum grande livro de métodos pedagógicos no qual se possa buscar qual a correta estratégia de ensino para alguma parte importante do currículo ou área de conhecimento.

Desta perspectiva o tutor pode ser visto como um sistema especialista ou resolvedor de problemas, e precisa ser construído em torno de sua estratégia de ensino. Em suma, para se poder construir "professores artificiais" é preciso primeiro descobrir como se ensina.

Considerando-se o ensino de programação observa-se diversos estilos de ensinar. Muitos grupos tem desenvolvido sua bem estabelecida cultura e uma série de preconceitos sobre o assunto. Geralmente as diferenças são baseadas no nível de importância que é dado a teoria e a prática, ou seja, qual o "melhor" caminho para adquirir conceitos de computação. Na opção "teoria antes da prática" desencoraja-se grandes quantidades de programação no

início, em favor de exercícios mais formais em algoritmos, estruturas de dados e lógica. Em sentido oposto, outros esperam obter base para o seu ensino na experiência prática de programação, que os alunos enfrentam o mais cedo possível.

Descendo mais um nível, mesmo dentro de cada um desses dois grandes estilos existem muitas "pedagogias" diferentes. Por exemplo, dentro da "experiência antes da teoria", muitos iniciam explorando sistematicamente as construções da linguagem escolhida; outros proveem, como caixas-preta, funções e procedimentos de alto nível e aos poucos vão mostrando como eles funcionam; e ainda outros baseiam seu ensino em construções básicas, como um laço que calcula uma somatória, e aos poucos mostram como podem ser utilizadas e alteradas para os mais diferentes propósitos.

Como pode ser observado ao analisar algumas implementações, as partes fortes e fracas dos sistemas são inerentes à cultura onde foram construídos.

Conhecimento sobre aprendizagem e ensino não estão ainda bem formalizados para que se possa decidir sobre qual seria o método

mais adequado. Acredito então que se deveria fazer opção por sistemas que não necessitassem ter uma metodologia implícita, ou seja, que fossem abertos de modo a se poder utilizá-los junto a qualquer estratégia de ensino. Este é o caso de alguns ambientes de suporte a serem discutidos na última parte deste capítulo e é a característica básica que norteou a definição do sistema que implementei neste trabalho.

Existem muitos sistemas tutores implementados e sendo amplamente utilizados. Alguns exemplos devem ser citados.

MALT [Koffmam, 1975] foi um dos primeiros tutores implementados e trabalha com programas em linguagem de máquina. Sua estratégia de ensino é bastante simples. Ele basicamente gera problemas e faz comentários sobre respostas dos estudantes, que são dadas em linguagem de máquina.

O sistema tem definidas duas ações de ensino. Uma é a de gerar um novo problema e apresentá-lo ao estudante, já subdividido em partes e a outra é a de fazer comentários sobre a solução apresentada pelo aluno para cada uma das partes.

O modelo do estudante consiste de marcas dentro da gramática do problema indicando o nível de "expertise" dentro de cada sub-problema e mais um parâmetro global que indica o nível geral de "expertise".

Neste sistema nota-se muita ênfase a sintaxe e semântica da linguagem em si, dando como pré-definida a parte de solução do problema para que possa ser implementado em linguagem de máquina. É questionável se, sem o auxílio do sistema, o aluno conseguirá fazer um programa a partir do problema.

TRILL [Cerri, 1984] é um tutor para um subconjunto de funções da linguagem Lisp. Ele trata somente as funções primitivas do Lisp e não as funções definidas pelo usuário.

Este sistema representa seu conhecimento sobre Lisp dentro de uma rede semântica e o treinamento do estudante é feito via a formulação de questões previamente armazenadas nos nós da rede. Se o estudante responde uma questão incorretamente, o sistema usa a rede para determinar quais conceitos são necessários para responder a questão. O sistema então atravessa cada um desses conceitos formulando questões ao usuário de modo a poder

localizar a fonte da falha do estudante. Este processo, se necessário, é aplicado recursivamente a sub-conceitos até que o sistema detete o ponto onde o primeiro erro ocorreu. Este modo de lidar com respostas incorretas é o aspecto mais interessante do sistema.

A rede semântica representa o conhecimento de um perito, e conforme o sistema vai percorrendo constrói um retrato do que o estudante conhece ou não, e isto é utilizado para determinar futuras questões. O modelo do estudante utilizado é o de sobreposição e portanto o sistema não consegue lidar com falhas de conhecimento não previstas ou então, formas diferentes de conhecimento.

GREATERP [Anderson, 1984; Anderson, 1985] é o mais expressivo sistema atualmente implementado.

Greaterp mantém um modelo procedural de um perito e de um iniciante em programação, representados sob a forma de regras de produção. As regras relativas aos principiantes representam os erros típicos que são observados em programas de principiantes. As regras relativas ao perito são extremamente detalhadas,

permitindo que o sistema resolva por si só os problemas propostos aos estudantes.

A estratégia do sistema é olhar o estudante bem de perto e interferir quando ele comete um erro.

A atividade do estudante quando interage com o sistema é construir um programa LISP. No nível mais alto, a interação entre o tutor e o estudante consiste de duas partes: o tutor propõe uma tarefa de programação e o estudante responde teclando o código.

O tutor tem uma base de conhecimento consistindo de diversas centenas de regras de solução de problemas as quais resolvem o problema, e outras tantas centenas de regras incorretas, representando erros de programação típicos de principiantes. Todo momento que o aluno dá um passo em direção à completar seu programa, ou seja, escreve um novo trecho, o tutor tenta mapear este passo em uma de suas regras. Se a regra que combina é uma das corretas, o tutor não interfere. Caso seja uma incorreta, o tutor imediatamente explica o erro. Se o estudante persiste errando o tutor pode mostrar qual é o passo correto, ou chamar o modo de planejamento, no qual o tutor mostra o algoritmo a ser

programado. O tutor reage a cada átomo LISP que é teclado e ao aluno só é permitido prosseguir com a resposta correta esperada pelo sistema.

Greaterp adota duas ações de ensino: sugere um problema para o aluno (possui um gerador de problemas) e faz comentários sobre os símbolos que o estudante coloca incorretamente, no momento em que é colocado, obrigando o aluno a adotar o caminho correto para poder prosseguir. O sistema é baseado na teoria de aprendizagem que sugere que os erros devem ser comentados tão logo possam ser detetados. Isto dá ao sistema um estilo autoritário de tutor, o que é justificado pelos autores ao argumentarem que não pode ser permitido ao estudante persistir em seus erros, pois com isso pode-se levar ao aprendizado dos erros ao invés do método correto. Isto em muito contrasta com teorias que argumentam que cometer erros, observar suas consequências e refletir sobre o que levou ao erro, são aspectos importantes, quando não cruciais, do processo de aprendizagem.

3.3 DEPURADORES INTELIGENTES ("BUG FINDERS")

Todos os tutores tem o problema de decidir se um estudante

respondeu corretamente uma questão ou apresentou a solução correta de um problema. Quando o objetivo é o ensino de programação, um sistema não decide facilmente se um programa preenche todos os requisitos do problema proposto, devido a dificuldade de descrever a resposta correta, no nível correto de detalhe e precisão e englobando todas as variações possíveis.

Os sistemas denominados depuradores, na sua maioria se concentram sobre erros de lógica, ao invés de sintáticos e geralmente possuem um editor de estruturas de forma a minimizar os erros sintáticos na escrita de programas. Estes sistemas adotam um dos três métodos descritos a seguir para isolar e detetar "bugs".

1. Casamento com uma resposta pré-definida

Existe definido no sistema uma resposta correta para cada problema e é recebida como entrada a resposta do aluno para o problema. São feitas transformações em ambos os códigos de modo a ter uma correspondência entre ambos e poder detetar eventuais erros. A principal dificuldade com este método é que geralmente o sistema não conhece os objetivos dos programas analisados e

portanto, não sabe o que é suposto que o código faça de modo a ter um melhor entendimento do que são os erros.

Um sistema que adota este método de deteção de erro é o **Laura** [Adam, 1980], que trata um programa do estudante em Fortran e um programa pré-especificado, e tenta verificar se correspondem efetuando uma série de transformações. Diferenças irreconhecíveis são tratadas como possíveis erros.

2. Casamento de acordo com a especificação

Neste método de deteção de erros o sistema parte de uma descrição em alto nível dos objetivos do código do estudante e verifica se o programa apresentado atende estes objetivos.

Este método, de modo geral, pode efetuar comentários mais pertinentes, devido ao conhecimento que tem sobre o que é esperado que o programa faça.

O sistema **Mycroft** [Goldstein,1975] deteta e repara erros em programas gráficos Logo através deste método. Ele constrói um programa a partir de um conjunto de asserções sobre as

propriedades geométricas do desenho a ser produzido. Ele deteta inconsistências entre o programa apresentado pelo estudante e as asserções utilizando teorias de planejamento de programas e erros em planos, e então corrige o programa.

O sistema espera que o usuário decomponha o problema de maneira a ter uma relação entre os subprocedimentos que serão escritos e a forma de decomposição. Consequentemente, é forçada uma forma estruturada de escrever o programa e o sistema não atua bem com programas mal estruturados, mesmo que estejam corretos.

Mycroft é um sistema de depuração bastante potente, mas o contexto da parte gráfica do Logo dá a ele uma série de vantagens e torna as técnicas que utiliza praticamente dependente deste contexto.

Proust [Johnson, 1985] é um sistema que tem a capacidade de detetar uma ampla variedade de erros lógicos em programas de estudantes escritos em Pascal. A idéia básica é a de identificar um erro através de uma biblioteca de erros. **Proust** recebe de entrada um programa tipicamente incorreto mais sua especificação e tenta entender o programa incorreto.

Proust tem uma série de diferentes estruturas de conhecimento: uma biblioteca de erros, uma biblioteca de planos de programação, conhecimento sobre objetivos do programador, etc. Seu procedimento inferencial tenta desenvolver uma estrutura de objetivos para o problema, a qual leva em conta todas as partes do código dentro do programa observado e usa tanto o plano correto como o incorreto, se necessário, como pontes entre uma particular decomposição do objetivo e o código falho. Proust trata a hipótese sobre as intenções do programador e sobre como ele tentou realizá-las, isto é, ele gera um apanhado de como o programa falho foi criado.

A essência do sistema Proust é seu sofisticado sistema de "matching" que faz a melhor combinação entre o programa de um estudante e uma estrutura esperada de objetivos e planos. Como os programas de estudantes geralmente não são corretos, este casamento nunca é exato tendo então que fazer uso de expectativas, derivadas da experiência sobre o modo como estudantes deformam planos, omitem objetivos, incluem objetivos extras ou incorretamente inter-relacionam planos. O sistema escolhe a combinação que minimize o número de violações da

hipótese e então as relata para o estudante, cobrindo uma grande variedade de problemas.

3. deteção de erros via diálogos

Este método foi explorado por Shapiro em [Shapiro, 1982] em seu sistema para programas em Prolog. Este sistema depende do fato de que um programa seja composto por uma hierarquia de chamadas de procedimentos que não possuem qualquer efeito colateral. O sistema executa o código apresentado pelo usuário e constrói uma árvore da sequência das chamadas dos procedimentos envolvidos. O sistema então examina esta árvore e faz perguntas sobre o comportamento que o usuário desejava e o comportamento que os procedimentos efetivamente estão tendo. Compara estes dados determina quais procedimentos estão errados.

Este método torna-se efetivo quando utilizado com linguagens tipo Prolog, que por serem de natureza declarativa tornam mais fácil a escolha de como relacionar mudanças no código com as mudanças desejadas no comportamento do programa. Não é claro se métodos com estas características podem ser aplicados à

linguagens procedurais, onde estas relações são difíceis de serem estabelecidas.

Por exemplo, a ausência de efeitos colaterais dos procedimentos remove a necessidade de distinguir entre passos de inicialização e sequências cruciais do problema, como é o caso do sistema **Mycroft** para a linguagem Logo.

3.4 AMBIENTES DE SUPORTE

Não existem linhas que caracterizam um sistema como ambiente de suporte, a não ser os objetivos para os quais foram criados e o modo de uso proposto. Portanto a melhor maneira de se definir um sistema desta categoria é através da descrição de implementações categorizadas como tal.

3.4.1 Bip [Barr, 1976]

Foi um dos precursores de sistemas que proporcionam ambientes que facilitem à atividade de programação para principiantes. Foi projetado para a linguagem Basic e é constituído de um

interpretador com facilidades gráficas de rastreio e uma série de problemas que compoem um currículo. Possui um mecanismo de seleção de tarefas que considera a performance do estudante, o currículo de problemas e o conjunto de habilidades que se deseja sejam aprendidas pelo aluno. O sistema tem um esquema muito pobre de verificação das respostas, fazendo uma simples comparação com a resposta esperada e quase nenhuma capacidade de identificar erros lógicos. As facilidades de suporte à programação incluem a habilidade de apresentar um "display" gráfico, passo-a-passo, do modelo de resposta esperado, um rastreio gráfico e "display" do valor das variáveis do programa do usuário.

Pode ser observado que o sistema em nada altera o processo de fazer programação e que sua característica mais forte é o rastreio gráfico, onde são ressaltadas características da máquina notacional Basic.

3.4.2. Bridge [Bonar, 1986]

É um sistema que, ao contrário do sistema Bip, tenta mudar o processo de fazer programas, ao invés de simplesmente facilitá-lo. É baseado na análise feita pelo sistema Proust, usando

objetivos e planos no processo de construir programas.

O sistema possibilita o desenvolvimento interativo do código através de três fases, onde o sistema apresenta sugestões quando solicitado. Na primeira fase, é apresentado um problema ao usuário e solicitado que ele construa uma sequência de objetivos como solução ao problema. O problema é apresentado em Inglês e os objetivos são descritos por frases também em Inglês, que são selecionadas pelo usuário a partir de um menu de frases. As frases partem do genérico e permitem uma elaboração mais específica até que correspondam, em nível de detalhe, à descrição do modelo de resposta esperado pelo sistema. Somente quando isto é conseguido é que o estudante passa para a próxima fase.

Na segunda fase, o usuário deve associar, a partir de um menu, planos a cada um dos objetivos especificados, que indicam o método geral de atingir os objetivos. No estágio final, o usuário constrói peças de código de programa que executam os planos, e gradativamente constrói, com ajuda de um editor de estruturas, o programa final.

Portanto, ao utilizar o sistema **Bridge**, o usuário obtém uma

série de representações declarativas da solução do problema original: a sequência de objetivos, a sequência de planos e o código em Pascal.

Um dos problemas deste sistema é o crescimento do número de frases a cada introdução de um novo problema. Outro problema observado pelos autores quando da utilização do sistema, foi a dificuldade que os estudantes tem em corretamente selecionar um plano a partir de um objetivo e a partir daí selecionar o respectivo código Pascal. Se este mapeamento não é tão óbvio para principiantes significa que o sistema em todas as suas etapas não conseguiu diminuir a barreira que existe entre algoritmos computacionais e algoritmos escritos em lingua natural para pessoas.

3.4.3. Tinker [Lieberman, 1987]

É um ambiente de programação criado com o objetivo de permitir que iniciantes adquiram mais rapidamente habilidades de programação na linguagem LISP.

A criação do sistema partiu do princípio teórico de que uma

das melhores formas de aprendizado é o aprendizado por exemplo, e que o uso de exemplos em programação pode trazer os mesmos benefícios do uso de exemplos no ensino de outras áreas de conhecimento.

Segundo o autor, não apenas a aprendizagem por exemplos é melhor para o estudante, mas para o professor, ensinar através de exemplos é mais fácil que ensinar apresentando princípios abstratos. E programar pode ser visto como o processo de ensinar o computador a executar um determinado procedimento.

O fato de exemplos terem um papel importante no ensino e esta analogia entre ensinar e programar, levou a implementação do sistema Tinker com o objetivo de verificar a hipótese de que, usar exemplos em programação traz os mesmos benefícios de usar exemplos no ensino.

O programador iniciante apresenta ao sistema diversos exemplos, ressaltando os aspectos essenciais e acidentais. O programador mostra como manipular os exemplos específicos e o sistema formula o procedimento para tratar o caso geral. Tinker então constrói um programa a partir da demonstração de exemplos,

usando os seguintes princípios do aprendizado por exemplos:

- para aprender a partir de exemplos, é necessário conhecer quais características são essenciais nos exemplos. Portanto, ao apresentar exemplos ao sistema, o programador precisa indicar quais aspectos ou constantes devem ser generalizados.

- exemplos, os quais mostram similaridades e diferenças entre uma idéia e idéias relacionadas, ajudam a clarificar os princípios. Tinker, então constrói um código condicional a partir do momento que o usuário apresenta um exemplo para cada caso importante no programa final. Quando mais de um exemplo é dado para uma função, Tinker pede ao usuário que informe qual é o teste que distingue o mais recente exemplo dos demais. Isto é análogo a maneira como programadores fazem testes em programas contendo condicionais, usando um exemplo para cada um dos casos de teste.

- a seqüência de exemplos deve começar com um exemplo simples até construir os exemplos mais complexos e casos excepcionais. A construção de procedimentos recursivos é feita no Tinker desta forma. Inicia-se definindo um procedimento o qual trata apenas o

caso base. A partir daí um exemplo do caso recursivo pode fazer uso do já definido procedimento para o caso base, e tendo os dois exemplos o sistema constrói o procedimento geral , com um teste que distingue ambos os casos.

Tinker é um sistema que possibilita programação instantânea, ou seja, a medida que se escreve os comandos, via um editor de funções selecionadas por menu, estes são imediatamente executados. Ao final da edição de todos os comandos e obtenção do resultado desejado, encerra-se a edição dando-se um nome à sequência de comandos.

Este sistema mostra uma tendência atual dos ambientes de programação no sentido de liberar o usuário do aprendizado de construções de uma dada linguagem de programação. Observa-se que toda enfase é dada à parte de solução de problemas e no "significado" dos conceitos quando utilizados para resolver problemas. Isto fica muito claro quando se observa a maneira como é feita a construção de procedimentos recursivos; não existe maneira de definí-los sem o entendimento do que significa recursão. O que se está observando é uma mudança no espectro do que era entendido como escrever um programa. Não se tem mais a

forma textual e sequencial de escrita, procurando-se formas mais próximas da maneira como usuários se expressam e resolvem problemas. A mesma tendência poderá ser observada no sistema Boxer descrito a seguir.

3.4.4. Boxer [diSessa e Abelson, 1986]

Boxer é um ambiente de programação altamente interativo especificamente projetado para tornar programação uma atividade fácil de aprender e comum à maioria das pessoas.

São utilizadas as seguintes características de um meio computacional facilitador ao aprendizado de programação:

- ENTENDIBILIDADE - característica básica para permitir o amplo acesso ao sistema, em termos de usuários não especializados.

- FAMILIARIDADE - se um meio é para ser útil e amplamente utilizado ele deve ser "familiar". Por exemplo, baseando estruturas de dados em textos ou gráficos, ao invés de fazer uso somente de estruturas altamente abstratas como "arrays" e listas.

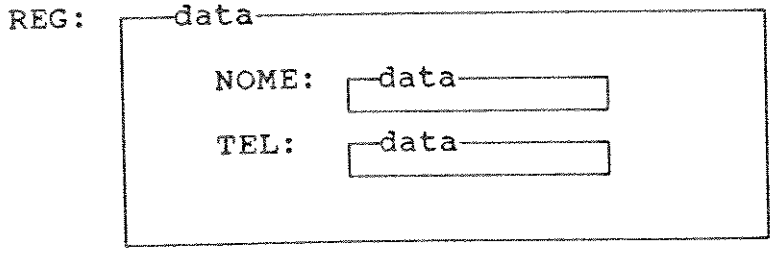
- DIRIGIDO PARA PEQUENAS TAREFAS - a habilidade de implementar facilmente idéias simples é muito mais importante nestes contextos que a habilidade de implementar eficientemente tarefas complexas.

- INTERATIVO - deve ser altamente interativo e não considerar a interface com o usuário em separado da estrutura e semântica da linguagem de programação.

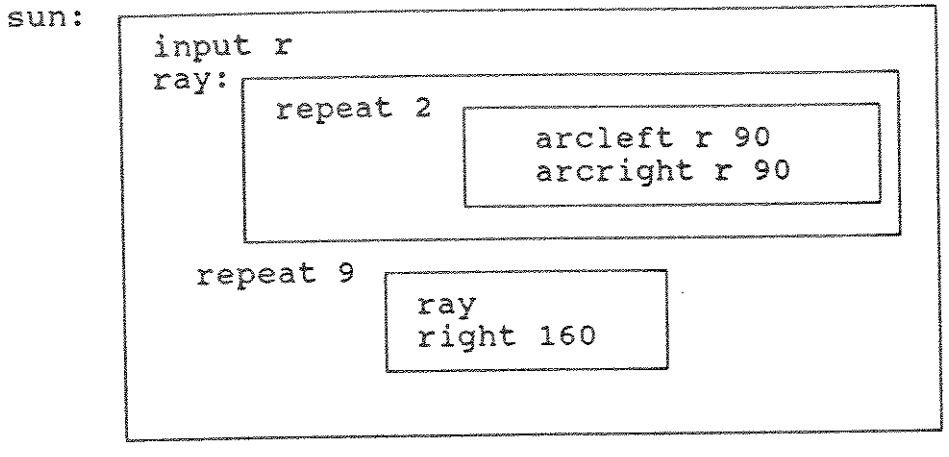
As características do sistema Boxer são determinadas por dois princípios chave: metáfora espacial e realismo ingênuo ("naive realism").

A metáfora espacial leva as pessoas a interpretar a organização do sistema computacional em termos de relações espaciais. Usar o sistema Boxer é como se mover em um amplo espaço bidimensional. Todos os objetos computacionais são representados em termos de caixas ("boxes"), as quais são regiões da tela que podem conter textos, gráficos ou outras caixas. Caixas dentro de caixas representam estruturas hierárquicas. Exemplificando:

- uma variável é uma caixa contendo o valor da variável. Para estruturas compostas, como registros formado por campos, a variável contém campos que são outras variáveis. Em Boxer teriamos:



-um programa é uma caixa contendo o texto do programa; sub-procedimentos e variáveis internas são representados como sub-caixas. Por exemplo:



Este é um programa em Boxer que faz desenhos utilizando os comandos gráficos da Tartaruga da linguagem Logo. Pode-se dizer, que de algum modo, Boxer é uma extensão da linguagem Logo, e portanto procedimentos simples de Boxer, especialmente os gráficos, se assemelham a procedimentos Logo. Observando o procedimento denominado SUN, verifica-se que ele tem um parâmetro de entrada denominado r ("input r") e um procedimento interno denominado RAY. As estruturas repetitivas também aparecem dentro de caixas, para ter claro quais comandos são repetidos.

As regras de escopo do Boxer somente possibilitam que se tenha acesso as definições e valores de uma caixa quando se está "dentro" dela. Tem-se portanto neste aspecto o uso da metáfora espacial "dentro de", representando as regras computacionais de escopo.

Quando se entra em uma caixa (basta colocar o cursor dentro da caixa) tem-se acesso ao seu conteúdo. Uma caixa portanto pode ser vista como definindo um ambiente com seus próprios dados e comportamento específico. Isto é reforçado pelo fato das caixas serem estritamente hierárquicas, e cada caixa existir

precisamente em um único lugar. Portanto um objeto do sistema Boxer é parte apenas de um outro objeto e existe uma única visão do objeto dentro do sistema, que é a provida pelo contexto espacial no qual o objeto existe (o ambiente em que é definido o objeto).

O realismo ingênuo é uma extensão da idéia "o que se vê é o que se tem" muito utilizada em editores de texto e planilhas eletrônicas, mas não em linguagens de programação. O ponto é que os usuários precisam acreditar que o que eles veem na tela é a totalidade do mundo computacional. Exemplificando:

- qualquer texto que aparece na tela - seja mensagem do sistema, escrito pelo usuário ou produzido por um programa - pode ser movido, copiado, modificado ou ser avaliado, caso seja o texto de um programa.

- pode-se alterar o valor de uma variável simplesmente alterando o conteúdo em tela, da caixa relativa àquela variável. Se um programa altera o valor de uma variável, o conteúdo da caixa respectiva é automaticamente atualizado.

As caixas gráficas permitem que o usuário faça qualquer desenho e armazene seu conteúdo. Estas caixas mostram as saídas dos procedimentos gráficos.

A primeira vista este esquema de caixas, pode parecer as usuais janelas utilizadas em sistemas computacionais interativos. Janelas entretanto, não tem qualquer significado computacional, exceto como lugares de comunicação entre o usuário e o sistema. As caixas do sistema Boxer, ao contrário, são os objetos computacionais do sistema, e os seus conteúdos refletem significados tais como sub-procedimentos, campos de registros, variáveis simples, etc.

De modo geral, o sistema Boxer desafia a visão atual de linguagens de programação, e consequentemente desafia a idéia do que programação deve ser e os objetivos que devem ter as linguagens de programação a serem criadas. Resta saber se esta nova abordagem possibilitará uma melhor compreensão e facilidade de programação.

3.5 CONCLUSÃO

Dois aspectos devem ser ressaltados quanto à utilização de sistemas computacionais no ensino de programação. O primeiro deles, é a preocupação subjacente a todos os sistemas de tentar alterar a forma convencional de ensino, de maneira a torná-la mais individualizada e motivadora. E isto vem sendo feito não com o intuito de mecanizar uma atividade tediosa, e sim de facilitar o acesso a uma atividade intelectual muito complexa. O segundo aspecto, é que o uso de uma nova ferramenta não introduz necessariamente mudanças metodológicas essênciais. E isto acontece por não se ter estabelecida a melhor metodologia a ser adotada no ensino de programação.

Dada esta indefinição e desconhecimento sobre qual o melhor método, ou então qual a melhor forma de programar, o que se tem são sistemas que carregam as qualidades e defeitos de uma determinada cultura computacional. Um bom exemplo é o sistema **Greaterp**, que provê um ambiente tutorial bastante claro, mas que provavelmente é muito ditatorial para alguns outros meios de ensino, outros professores e outros alunos, que definam uma outra cultura.

Tendo estas dificuldades em vista, acredito ser muito pouco viável, pelo menos a curto prazo, que um sistema que tente forçar e definir uma metodologia de programação e, consequentemente, uma metodologia de ensino tenha amplo sucesso. Com isto, acredito que o caminho para a utilização do computador como auxiliar no aprendizado de programação, seja o de sistemas abertos que não tenha o objetivo de ensinar e sim de melhor subsidiar à atividade de programação.

Dentre os sistemas apresentados deve-se destacar o sistema Tinker e o sistema Boxer por apresentarem uma nova abordagem da atividade de programação. Parecem, a princípio, ser o tipo de ambiente mais prometedor, por tentarem ser próximos ao que é feito usualmente por pessoas e, consequentemente, diminuir as barreiras de acesso ao computador. Além disso são abertos o suficiente para serem utilizados dentro de qualquer estratégia de ensino de programação que venha a ser adotada.

Estes ambientes procuram facilitar a atividade de programação a medida em que objetivam maneiras mais efetivas de espelhar os processos mentais dos usuários. Fornecem um tipo de interação que fornece ao principiante ferramentas para estruturar este novo

domínio de conhecimento.

Estas idéias de que o computador não é para ensinar coisa alguma, sendo apenas uma riquíssima ferramenta para auxiliar a pensar e a expressar pensamento, está presente na metodologia de uso do computador no aprendizado, definida pela "cultura Logo". E foi partindo dessas premissas básicas que defini o tipo de sistema que implementei, que tem como objetivo final instrumentar as pessoas, de forma a que possam pensar em programação, enquanto um processo executado por uma máquina. Implicitamente está-se interessado em entender como se programa, para daí entender como facilitar este aprendizado.

No próximo capítulo deste trabalho será apresentada a linguagem Logo que possui características, tanto como linguagem quanto como metodologia de ensino de programação, também presentes em ambientes como Tinker e Boxer.

Capítulo 4

Logo: Linguagem e Metodologia

4.1 Introdução

Logo é uma linguagem de programação projetada com o objetivo de facilitar o aprendizado de programação, sendo bastante simples de ser assimilada e utilizada. Além disso ela possui características elaboradas para implementar uma metodologia de ensino baseada no computador e explorar aspectos do processo de aprendizagem, denominada Metodologia Logo.

O objetivo deste capítulo é fazer uma apresentação do Logo, levando em conta estas diversas facetas, que acredito serem relevantes para o entendimento da problemática que gerou a pesquisa desta tese. Será portanto feita uma descrição de Logo como linguagem de programação e dos aspectos metodológicos relativos à apresentação de conceitos computacionais. Além disso, serão analisados os problemas no aprendizado de conceitos de programação na forma em que se dá o aprendizado de Logo.

4.2 Logo como Linguagem de Programação

4.2.1 Aspectos Gerais

Logo foi desenvolvida por volta de 1968, por um grupo de pesquisadores liderados pelo Prof. Seymour Papert. A intenção de Papert era prover um ambiente computacional que pudesse servir como um **rico ambiente de aprendizagem.**

Ao mesmo tempo Logo foi projetada quando o laboratório de Inteligência Artificial do M.I.T. começava a florescer. Por estar ligado ao laboratório, Papert utilizou muitas idéias e conceitos de Inteligência Artificial ao projetar Logo. Alguns dos conceitos emprestados por Papert do laboratório foram os que embasam a linguagem LISP. Outro conceito fundamental também emprestado, e através do qual Logo se tornou mais conhecida, é a parte de **"gráficos da Tartaruga".**

Em [McMillan, 1987] são levantados os problemas ao aprender linguagens ditas como sendo de Inteligência Artificial, especialmente LISP. Ele afirma que com "background" em outras linguagens tradicionais de programação, como Pascal ou Basic, é difícil aprender uma linguagem de computação simbólica como LISP.

A partir daí o autor sugere que se aprenda Logo como uma "ponte" para o aprendizado de LISP. Certamente esta sugestão causa espanto entre profissionais de computação, pois Logo tem sido vítima de seu próprio sucesso em escolas de primeiro grau, adquirindo com isso a reputação de uma linguagem para "bebes". É conhecida como a linguagem de "casinhas e quadradinhos" e de "programadores de cinco anos de idade". Nestes preconceitos tem-se clara a confusão entre simplicidade e trivialidade.

Logo, na verdade, é uma poderosa linguagem de programação para computadores pessoais e é uma linguagem para todas as idades de programadores.

O que significa uma linguagem ser poderosa?

Certamente não significa que se pode escrever programas em uma determinada linguagem, que produzam certos resultados, e que não poderiam ser escritos em outras linguagens. Neste sentido, todas as linguagens são iguais pois, se pode ser escrito um programa em Logo ele também pode ser escrito, de uma forma ou de outra, em Pascal ou Basic. E, é claro, o inverso também vale.

Ao invés disto, pode-se dizer que, o poder de uma linguagem está na medida de o quanto a linguagem ajuda o programador a se concentrar no problema real a ser resolvido, ao invés de ter que se preocupar com restrições da linguagem.

Por exemplo, em Pascal, Basic e em todas as linguagens originalmente derivadas do FORTRAN, o programador tem que ser muito explícito sobre o que acontece dentro da memória do computador. Se desejar agrupar vinte números como uma unidade, é necessário declarar um "array", dizendo-se de antemão que serão vinte números a serem agrupados. Se por acaso desejar utilizar o mesmo programa para agrupar vinte e um, o programa terá que ser alterado, informando que agora serão agrupados vinte e um números ao invés de vinte. É necessário dizer antecipadamente que vai ser utilizado um "array" de vinte números inteiros, ou vinte números reais ou então vinte caracteres e durante o programa, a quantidade de elementos e o tipo de elemento a ser agrupado não podem ser modificados.

Em Logo todo o processo de alocação de espaço é automático. Se o programa produz uma lista de vinte números, o espaço para esta lista é provido automaticamente. Se mais tarde desejar

adicionar o vigésimo primeiro ou o vigésimo segundo número, também é automático. Não existem restrições quanto ao tipo do elemento a ser agrupado, podendo ser alterado a qualquer momento, ou até misturado dentro de uma mesma estrutura.

Pode-se contra-argumentar, ressaltando-se que versões de linguagens como o Pascal permitem alocação dinâmica, o que resolve o problema de prever antecipadamente a quantidade de elementos a serem agrupados. Isto é real, mas ainda neste caso a alocação continua restrita a um único tipo de elemento a ser definido "a priori". Além disso, este tipo de alocação exige uma idéia clara do sistema de alocação de espaço pois trabalha diretamente com endereços de memória.

A liberdade no tratamento de variáveis pode ser vista como um aspecto negativo, a medida que não impondo uma disciplina pode resultar em programas muito difíceis de serem depurados e entendidos. Mas pelo contrário, o que tem sido observado é que a medida que as pessoas avançam na programação, naturalmente tendem a ter uma disciplina na criação e utilização de variáveis. O interessante disto é que não é necessária a imposição de uma disciplina através da linguagem a ser utilizada, o próprio

aprendizado e consequente amadurecimento dos conceitos de programação e computação conduzem a uma boa utilização de variáveis.

A disciplina de declarar tudo "a priori" advém da idéia de planejamento da solução de problema, implementando uma concepção de "engenharia" em programação. Programar implica em "projetar" o programa, daí a fase de planejamento (especificação), e depois "construção", etc. (programação estruturada). As linguagens que sustentam esta metodologia foram feitas por engenheiros e não por pessoas que tinham o aprendizado como objetivo principal. O Logo foi feito com intenção de facilitar a interação com a máquina e o aprendizado de programação.

Existem outras restrições da linguagem que muitas vezes conduz o principiante a focalizar mais atenção nas peculiaridades da linguagem de programação do que na solução do problema a ser programado. Um exemplo claro disto é a sintaxe da linguagem, ou seja, as regras para construir instruções válidas. Todas linguagens derivadas do FORTRAN tem uma dúzia ou mais tipos de instrução, cada qual com uma sintaxe peculiar. Por exemplo, o comando PRINT em BASIC requer uma lista de expressões que se

deseja imprimir. Se as expressões são separadas com vírgula, significa imprimí-las de um determinado modo; se separadas com ponto e vírgula significa impressão de outro modo. Mas não é permitido utilizar ponto e vírgula em outra espécie de comando que também requera lista de expressões. Em Logo existe apenas uma forma de sintaxe que é a de chamada de procedimentos.

Por que estas diferenças existem ?

Naturalmente não é simplesmente porque são linguagens diferentes, como é diferente o Português do Japonês. FORTRAN foi inventado antes que as bases matemáticas de programação de computadores fossem bem entendidas, portanto seu projeto em muito reflete as capacidades e deficiências dos computadores disponíveis na época. As linguagens baseadas no FORTRAN continuam conservando o mesmo projeto fundamental, embora alguns de seus piores detalhes tenham sido modificados nas versões mais recentes, como o Pascal por exemplo.

As linguagens atualmente tidas como mais poderosas são baseadas em algum modelo matemático particular de computação e usam este modelo de forma consistente. Por exemplo, APL é baseada

na idéia de manipulação de matrizes; Prolog, a linguagem famosa por ser dita de quinta geração, é baseada em cálculo de predicados, a forma de lógica matemática. Logo como LISP, é baseada na idéia de composição de funções.

Outro aspecto que deve ser considerado é os paradigmas subjacentes às linguagens de programação. Paradigmas, segundo Baranauskas em [Baranauskas, 1991] podem ser definido como diferentes modelos de representação de problemas a serem resolvidos pela máquina.

"Considerando as linguagens de programação sob seu aspecto de evolução cronológica, elas representam, em graus variados, abstrações da arquitetura subjacente, chamada Von Neumann. O paradigma procedural, é o que mais se aproxima do uso da arquitetura Von Neumann como modelo para representação de um problema a ser resolvido pela máquina. Segundo o paradigma procedural, programar o computador significa "dar-lhe ordens" que são executadas sequencialmente. Em tal paradigma, "representar" um problema para ser resolvido pelo computador envolve escrever uma série de ações (procedimentos) que executadas levam à solução." [Baranauskas, 1991]

Além do paradigma procedural, existem definidos o paradigma funcional, lógico e o orientado a objetos, que são definidos em [Baranauskas, 1991] da seguinte forma.

O paradigma funcional está baseado no uso de funções matemáticas como modelo para representação do problema a ser resolvido pela máquina. Segundo este paradigma, programar o computador significa definir funções, aplicar funções e conhecer o comportamento das funções. Assim definir um problema para ser resolvido num ambiente funcional necessita de uma abordagem diferente dos métodos usados em linguagens procedurais.

Sob o enfoque do **paradigma da programação em lógica**, representar um problema para ser resolvido pelo computador, consiste em expressar o problema na forma de lógica simbólica e utilizar o processo de inferência lógica para produzir resultados. "Os programas consistem de declarações, ou seja, de proposições assumidas como verdadeiras a respeito do domínio de um problema."

A idéia básica do **paradigma orientado** a objetos é imaginar que programas simulam o mundo real, ou seja, um mundo povoado de

objetos. Isto é consequência da idéia de que no mundo real frequentemente usamos objetos sem precisarmos conhecer como realmente eles funcionam. Assim, programação orientada a objetos pressupõe um ambiente onde múltiplos objetos podem coexistir e trocar mensagens entre si.

"Programar nos diferentes paradigmas significa, portanto, representar segundo modelos diferentes o problema a ser resolvido. Cada linguagem que suporta determinado paradigma representa, portanto, um meio onde o problema é resolvido" [Baranauskas, 1991]

No trabalho com Logo não se está restrito à apenas um paradigma. Tem-se o paradigma procedural, quando na parte gráfica de comandos à Tartaruga. O paradigma funcional aparece quando se está trabalhando com a parte de manipulação simbólica. E, finalmente, experimenta-se o paradigma orientado a objeto quando do trabalho com múltiplas Tartarugas ou "sprites".

Isto de certa forma enriquece o trabalho com Logo como primeira linguagem de programação, facilitando o uso futuro de outras linguagens restritas a um único paradigma. Ao mesmo tempo,

os problemas de mudança de paradigma são todos sentidos no trabalho com uma única linguagem e de certa forma agravados.

4.2.2 Descrição da linguagem Logo

Logo é uma linguagem de programação de propósito geral, ou seja, não foi projetada para resolver problemas de um domínio específico. Ela é interpretada, procedural e recursiva. O seu conjunto de comandos subdividem-na em duas partes básicas: gráfica (paradigma procedural) e de processamento simbólico (paradigma funcional) e gráfica com múltiplas Tartarugas (paradigma orientado a objetos). Esta subdivisão não é rígida no sentido de se estar no modo gráfico ou não, pois projetos em Logo geralmente envolvem comandos dos diferentes tipos simultaneamente. Através dos objetos simbólicos que Logo manipula, como palavras, listas, sentenças, listas de propriedades, etc., torna-se fácil representar leis de jogos, leis de Física, Química, etc., podendo-se elaborar programas com heurísticas de controle bastante sofisticadas.

Os comandos da parte gráfica manipulam uma "Tartaruga", que é representada na tela por um pequeno triângulo.

A "Tartaruga" na tela tem três estados:

-posição que identifica onde ela se encontra na tela, correspondendo às coordenadas X e Y do sistema cartesiano de coordenadas.

-direção que idendifíca para onde ela está apontando, correspondendo à coordenada Z do sistema polar de coordenadas.

-objeto em uso: lápis (deixa riscos ao se movimentar), borracha (apaga riscos ao se movimentar) ou nada, isto é, nem lápis nem borracha (ao se movimentar não deixa riscos e nem apaga riscos)

A "Tartaruga" responde comandos bastante simples:

.ParaFrente (PF) move a "Tartaruga" na direção em que está apontando um certo número de unidades denominadas passos. Teclando PF 50, a "Tartaruga" responde movimentando 50 "passos". Analogamente é definido o comando ParaTrás (PT).

.ParaDireita (PD) faz com que a Tartaruga efetue uma rotação no sentido horário um dado número de graus. Analogamente existe o comando ParaEsquerda (PE)

.UseLápis (UL), UseBorracha (UB) e UseNada (UN) alteram o objeto que a Tartaruga está usando.

.DesapareçaTat (DT) e ApareçaTat (AT) fazem com que ela desapareça e apareça na tela (o pequeno triângulo torna-se invisível ou visível).

.TAT deixa a tela totalmente limpa e a Tartaruga posicionada no centro da tela.

Exemplificando o uso dos comandos, para desenhar um quadrado

poderia ser escrito o seguinte conjunto de instruções:

```
pf 50
pd 90
pf 50
pd 90
pf 50
pd 90
pf 50
pd 90
```

ou, utilizando o comando **Repita** que faz a repetição incondicional, um certo número de vezes, dos comandos de uma lista de comandos:

```
repita 4 [pf 50 pd 90]
```

A lista de comandos a ser repetida é colocada entre [e], que é a notação de listas em Logo.

Logo permite a definição de procedimentos. Por exemplo, para definir um procedimento que desenhe um quadrado pode-se escrever:

```
aprenda quadrado
repita 4 [pf 50 pd 90]
fim
```

O comando **Aprenda** indica a definição de um procedimento com o nome **Quadrado** e **Fim** especifica o fim da definição do procedimento.

Uma vez definido, a palavra **Quadrado** passa a fazer parte do vocabulário da linguagem e, sempre que desejar desenhar um quadrado basta dar o comando **Quadrado**. Devido ao fato de todos os comandos de Logo possuirem a mesma sintaxe, qual seja, a de chamada de procedimento, facilmente é incorporada esta característica de extensibilidade da linguagem, pois não existirá uma diferença entre o modo de uso das primitivas da linguagem e dos procedimentos definidos pelo usuário.

Pode-se alterar o procedimento **Quadrado**, de forma que ele tenha uma entrada especificando o tamanho do quadrado a ser desenhado. Tem-se então a definição de um procedimento com parâmetros:

```
aprenda quadrado :tamanho
repita 4 [pf :tamanho pd 90]
fim
```

Para utilizar o procedimento assim definido basta efetuar a chamada:

```
quadrado 70
```

e na tela é desenhado um quadrado de lado com tamanho 70.

Não existem restrições quanto ao número de parâmetros de um procedimento e o tipo de passagem é sempre por valor.

Desde que procedimentos definidos passam a "incorporar" a lista de comandos da linguagem, pode-se utilizá-los dentro da definição de outros procedimentos, tendo-se então o conceito de sub-procedimento e passagem de valores. Por exemplo, pode-se definir:

```
aprenda desenho
repita 6 [pf 20 pd 60 quadrado 50]
fim
```

ou, utilizando parâmetros:

```
aprenda desenho :x
repita 6 [pf 20 pd 60 quadrado :x]
fim
```

É importante observar que a noção de subprocedimento em Logo é absolutamente conceitual. Dentro do Logo não existe a idéia de programa como uma unidade. Cada procedimento é uma entidade por si só. Um procedimento em especial pode ser pensado como o principal, mas o interpretador não tem este conhecimento; pode-se invocar qualquer procedimento diretamente através de uma instrução interativa de chamada de procedimento, desde que ele faça parte da área de trabalho. Em Logo a área de trabalho está restrita ao espaço de memória reservado para uso, e geralmente nela está presente uma coleção de procedimentos e variáveis

interrelacionadas logicamente, de acordo com a especificação feita pelo usuário.

Todas as variáveis definidas dentro da área de trabalho são globais, a menos das variáveis definidas como parâmetros ou então declaradas locais através da declaração LOCAL colocada no inicio da definição do procedimento. O escopo de validade das variáveis em Logo é dinâmico e não estático, ou léxico, como na maioria das linguagens estritamente procedurais.

Logo também permite a definição de procedimentos recursivos. Poderia-se então definir um procedimento recursivo que desenhasse um quadrado da seguinte forma:

```
aprenda quadrec  :tam
pf :tam
pd 90
quadrec :tam
fim
```

Este procedimento tem uma recursão infinita, devendo parar quando o usuário interromper. A possibilidade de definir

procedimentos recursivos acrescenta muito poder à linguagem. Um exemplo interessante é o procedimento **Poli** definido a seguir:

```
aprenda poli :passos :angulo
pf :passos
pd :angulo
poli :passos :angulo
fim
```

Ao chamar **Poli 20 120** tem-se desenhado um triângulo, **Poli 20 72** desenha um polígono de 5 lados, **Poli 10 45** desenha um polígono de 8 lados e **Poli 25 144** irá desenhar uma estrela de 5 pontas.

Para desenhar espirais bastaria incrementar a cada chamada recursiva o número de passos:

```
aprenda espi  :passos  :angulo
pf :passos
pd :angulo
espi :passos+5 :angulo
fim
```

Para não ter recursão infinita deve-se indicar uma condição de parada, através do comando condicional Se:

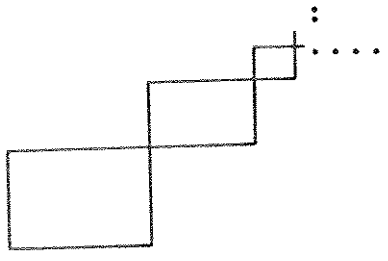
```
aprenda espi  :passos  :angulo  :parada
se :passos > :parada [pare]
pf :passos
pd :angulo
espi :passos +5   :angulo   :parada
fim
```

Procedimentos recursivos com operações no retorno podem ser exemplificados pelos procedimentos a seguir:

```
aprenda escada :num :tam
se :num < 1 [pd 180 pare]
degrau :tam
escada :num-1 :tam-5
degrau :tam
fim
```

```
aprenda degrau :tam
pf :tam pd 90
pf :tam pe 90
fim
```

que desenhará uma escada do seguinte tipo:



Como Logo não possui nenhum comando primitivo de repetição condicional, recursão é indispensável quando não se conhece "a priori" o número de repetições necessárias à solução de um problema.

O conjunto de ações de Logo é formalmente subdividido em duas categorias: COMANDOS e OPERAÇÕES.

COMANDOS são ações que sempre produzem um resultado explícito na tela, um desenho ou impressão de algo. Por exemplo, PF é um comando que resulta em um deslocamento da Tartaruga. ESC é um comando que resulta na escrita de um valor ou mensagem na tela. Estes implementam o paradigma procedural.

OPERAÇÕES são outros tipos de ações que não resultam num produto explícito na tela, mas sim na passagem de informação. Portanto operações retornam valores ou informação para serem utilizadoas por comandos. Isto implementa o paradigma funcional e portanto são de natureza diferente das funções nas linguagens procedurais.

Em Logo pode-se portanto também definir operações. Pode-se escrever:

```
aprenda média :x  :y
envie (:x + :y) / 2
fim
```

Assim definido, o procedimento **Média** toda vez que for chamado

retornará um resultado que poderá ser examinado diretamente ou então ser entrada de um outro procedimento. A indicação de que um valor está sendo retornado é feita através da utilização do comando **ENVIE**. Este é um comando especial que só pode ser utilizado dentro de procedimentos e nunca no modo direto. Ele necessita de uma entrada que pode ser qualquer objeto da linguagem. O efeito do ENVIE é fornecer o objeto dado a ele de entrada como saída da operação. O comando **ENVIE** é a "ponte" entre o paradigma procedural e o funcional, pois se Logo fosse estritamente procedural, como é o caso de linguagens como o Pascal, não necessitaria de um comando especial para denotar retorno de valores, que no caso sempre retornam através da atribuição de um valor a uma variável. O inverso também é verdade, ou seja, se Logo fosse estritamente funcional, como o caso do Lisp puro, não precisaria do **ENVIE**.

Além da parte gráfica, Logo possui um conjunto de operações para manipulação de símbolos.

Os objetos que Logo manipula simbolicamente são dois: **PALAVRAS** e **LISTAS**.

PALAVRA é uma sequência de caracteres precedida por aspas ("). Por exemplo: "casa, "123, "xpto20, "logo. NÚMERO é um tipo especial de palavra, constituída só de dígitos, podendo ser qualquer número inteiro ou real (com vírgula decimal). Por exemplo: 123, 345,5, 1,5, 46.

LISTA é a maneira de Logo manipular uma coleção de dados. É um conjunto de palavras ou listas entre colchetes ([]). Por exemplo:

[1 23 casa boneca bola]

é uma lista de 5 elementos, onde os dois primeiros são números e os demais palavras (ou todos palavras).

O elemento de uma lista também pode ser uma lista:

[[cachorro quente] hamburger pastel]

é uma lista de três elementos onde o primeiro é uma lista de dois elementos.

Existe a definição de LISTA VAZIA que é denotada por [] e PALAVRA VAZIA que é denotada por aspas (") seguida de branco.

Logo possui muitas operações para manipular listas. Todo o conjunto de operações pode ser subdividido em cinco grandes grupos, de acordo com suas funções:

Grupo 1 - PREDICADOS

Operações que retornam valores booleanos identificados pelas palavras VERD e FALSO, representando verdadeiro e falso respectivamente. São primitivos predicados que verificam se um determinado objeto é uma palavra, ou uma lista, ou elemento de uma lista, etc.

Por exemplo, ÉLISTA é um predicado que tem um parâmetro e que retorna "VERD se o parâmetro for lista e "FALSO caso contrário. Analogamente, existem os predicados ÉPALAVRA e ÉNÚMERO.

ÉVAZIA é um predicado com um parâmetro (palavra ou lista) e que retorna "VERD se o parâmetro for vazio e "FALSO caso

contrário.

SÃOIGUAIS tem dois parâmetros, palavras ou listas indistintamente. Retorna "VERD se os parâmetros são idênticos ou se ambos os parâmetros forem números e possuírem o mesmo valor numérico.

Em Logo todos os predicados primitivos começam com É ou SÃO, para indicar que se está fazendo uma pergunta com duas respostas possíveis: SIM e NÃO, que correspondem as palavras VERD e FALSO respectivamente.

Pode-se definir predicados. Por exemplo:

```
aprenda éímpar :número
envie sãoiguais resto :número 2 1
fim
```

que verifica se um número é ímpar ou não.

Grupo 2 - SELECIONAR ELEMENTOS

Basicamente existem dois tipos de operações que facilitam a separação de elementos de palavras e listas: as operações que retornam o primeiro ou o último elemento de uma lista ou palavra (PRI e ULT) e as operações que retornam o restante de palavras e listas sem o primeiro ou sem o último elemento (SP e SU).

Por exemplo,

pri [1 23 casa boneca bola] retorna o número 1

ult [1 23 casa boneca bola] retorna a palavra bola

pri "cachorro retorna o carácter c

ult "gato retorna o carácter o

sp [123 casa boneca bola] retorna a lista [casa boneca bola]

su [1 23 casa boneca] retorna a lista [1 23 casa]

```
sp "bola     retorna a palavra ola

su "bola     retorna a palavra bol
```

Pode-se combinar operações, obtendo operações mais complexas. Por exemplo:

```
pri sp [[cachorro quente] hamburger pastel]

                    retorna a palavra  hamburger
```

Grupo 3 - BUSCAR E CONTAR ELEMENTOS

Para selecionar um elemento de uma determinada posição em uma lista ou palavra existe a operação ELEMENTO, que tem dois parâmetros, o primeiro indicando a posição desejada e o segundo o objeto (palavra ou lista). Por exemplo:

```
elemento 2   [casa [pneu carro]] retorna a lista [pneu carro]
```

elemento 5 "preto retorna o carácter o

A operação **Num.ELem (NEL)** retorna o número de carácteres de uma palavra ou o número de elementos de uma lista, dados como parâmetro. Por exemplo:

nel [[abc]] retorna 1

nel "abc retorna 3

Grupo 4 - FORMAR ESTRUTURAS

São definidas operações que constrõem estruturas a partir de duas ou mais entradas. As estruturas que podem ser montadas são palavras através da operação PALavra (PAL) e listas a partir da operação LISTA. Por exemplo:

pal "gato "preto retorna a palavra gatopreto

lista "gato "preto retorna a lista [gato preto]

Grupo 5 - CONCATENAR ELEMENTOS

As operações JuntenoFim (JF) e JuntenoInício (JI) permitem que se adicione elementos no final ou início de uma lista. Precisam de dois argumentos e retornam uma lista que é a junção dos dois argumentos. Por exemplo:

```
jf [coca] [[cachorro quente]  hamburger  pastel]
```

retorna a lista:

```
[[cachorro quente]  hamburger pastel [coca]].
```

```
ji "coca [[cachorro quente] hamburger pastel]
```

retorna a lista:

```
[coca [cachorro quente] hamburger pastel]
```

SeNtença (SN) é uma operação similar á operação JI, no sentido de formar uma única lista a partir de seus argumentos. A diferença é que SN tem diversas listas como entrada e combina seus elementos de modo a produzir uma única lista. Assim, o comando:

sn [cachorro quente] [é mais gostoso]

retorna a lista [cachorro quente é mais gostoso] e,

ji [cachorro quente] [é mais gostoso] retorna

a lista de dois elementos: [[cachorro quente] [é mais gostoso]]

Um aspecto significante a respeito da estrutura de lista em Logo é que ela pode ser manuseada como objeto de primeira classe, ou seja, pode ser atribuída à variáveis, passada como parâmetro e retornada como valor de operações. Por exemplo, pode-se escrever:

coloque [casa arvore] "x

que é uma das formas do comando de atribuição em Logo, significando que se está atribuindo como valor à variável de nome x a lista [casa árvore]. Em Logo se diferencia sintaticamente a referência ao nome ou valor de uma variável. Quando se deseja referenciar o nome deve-se preceder a variável de " e quando se deseja o valor deve-se preceder de :. Assim "x referencia a palavra x ou então o nome da variável x, e :x referencia o valor da variável de nome x. Por exemplo:

```
esc "x          escreve a palavra x

esc :x          escreve o conteúdo da variável de nome x
```

Um dos procedimentos básicos que deve ser escrito é o de percurso de palavras ou listas:

```
aprenda percorrer :estrutura
se évazia :estrutura [ pare ]
esc pri :estrutura
percorrer sp :estrutura
fim
```

Se quizéssemos escrever um predicado que verificasse se um elemento é ou não membro de uma lista ou palavra, poderíamos escrever:

```
aprenda émembro :elemento :estrutura
se évazia :estrutura [envie "falso]
se pri :estrutura = :elemento [envie "verd]
envie émembro :elemento sp :estrutura
fim
```

Para contar o número de elementos de uma lista ou palavra, pode-se definir:

```
aprenda numelem  :estrutura
se évazia :estrutura [ envie 0 ]
envie 1 + numelem sp :estrutura
fim
```

Para inverter uma lista:

```
aprenda inverta :lista
se :lista = [] [envie []]
envie jf pri :lista inverta sp :lista
fim
```

Para fazer qualquer tipo de procedimento interativo, é necessário manipular estruturas na forma de listas ou palavras. Logo possui duas operações de entrada: LINE e CARE. A primeira retorna uma lista de tudo que for teclado até o return e a segunda o primeiro caracter teclado. Um procedimento interativo bastante simples pode ser exemplificado por:

```
aprenda diálogo
esc [Benvindo ao Sistema]
esc [É a primeira vez que voce o utiliza (s/n)]?
se care = "n [esc [Bom te ver de novo!]] [ esc [ Prazer em
recebe-lo pela primeira vez !]]
esc [Qual é o seu nome?]
esc (sn [Voce quer jogar xadrez] pri line [?(s/n)])
se care = "s [xadrez]
fim
```

Esta são as características básicas de Logo como linguagem de programação. Na próxima seção deste capítulo são abordados aspectos relevantes da metodologia de ensino que permeia o uso da linguagem Logo no aprendizado de programação.

4.3 Metodologia Logo

Os aspectos metodológicos que envolvem a utilização da linguagem Logo em um ambiente de aprendizagem são fundamentais para que se possa compreender quais características deve ter qualquer extensão computacional do ambiente e em que pontos o ambiente pode ser fortalecido de modo a propiciar o alcance dos objetivos a que se propõe.

Dentro da abordagem que farei da metodologia tratarei a forma que conceitos computacionais se apresentam no decorrer da utilização da linguagem, bem como de outros conceitos, não necessariamente computacionais, que estão envolvidos na utilização da linguagem.

Sempre a introdução ao Logo é feito através de sua parte gráfica, que é denominada "porta de entrada". Isto é feito devido ao fato de que os problemas a serem resolvidos graficamente exploram atividades espaciais intuitivas e por possuir um conjunto de comandos de terminologia fácil e com ações bem definidas. As atividades gráficas desenvolvidas inicialmente são bastante simples e envolvem conceitos espaciais que são adquiridos nos primórdios de nossa infância, quando começamos a engatinhar, e que geralmente permanecem a nível intuitivo, sem nenhuma formalização.

Os comandos da Tartaruga são termos usados no dia a dia. Dada esta analogia, utiliza-se o conhecimento intuitivo de nosso corpo e de como ele se movimenta, formalizando-o através de comandos que movimentam a Tartaruga. Um aspecto interessante observado nesta fase inicial do aprendizado é que muitas vezes pessoas conseguem ir de sua casa a algum outro lugar, mas não percebem que estão usando conceitos como distância, ângulo reto para dobrar esquinas, etc. No comando da Tartaruga estes conceitos precisam ser explicitados, e com isso obtem-se o desenvolvimento, ou formalização de conceitos espaciais e geométricos, uma vez que são exercitados, depurados e utilizados em diferentes situações.

Quase sempre a construção do quadrado é a primeira sequência de instruções que uma pessoa que é apresentada ao Logo constrói.

Parece trivial, mas observa-se que mesmo adultos não ficam a vontade com estas construções geométricas simples. Esta dificuldade pode ser exemplicada através da construção do triângulo. Inúmeros adultos descrevem o triângulo equilátero da seguinte forma:

```
repita 3 [pf 50 pd 60]
```

Estou enfatizando os adultos por ser previsto que eles de alguma forma já formalizaram estes conceitos. Mas isto é válido para a grande maioria de usuários que temos observado. Qual não é a surpresa quando da execução do comando, o triângulo não fecha. Observa-se então o conhecimento estanque que é adquirido na escola de que "o triângulo equilátero tem todos os lados iguais e os ângulos de 60 graus". Mas, ao comandar a Tartaruga é preciso pensar em ângulo externo e interno e na relação entre eles e, então, descrever o triângulo como:

```
repita 3 [pf 50 pd 120]
```

Observe que em um exemplo extremamente simples como este, explora-se diversos conceitos tais como, diferença entre deslocamento e giro, ângulos externos e internos, distância, lateralidade, sequência lógica de execução de instruções, etc. São conceitos que no ensino formal são abstratos e no Logo palpáveis e concretos a partir do momento que são utilizados em algo prático.

Inicia-se portanto, explorando o paradigma procedural de "dar ordens" por ser mais fácil de ser compreendido. Um programa é visto como uma sequência de "ordens a um animal com comportamento bem definido".

Outra dificuldade é conseguir chegar à construção de uma circunferência. A primeira reação é de que é impossível pois "a Tartaruga não sabe andar em curva ela só sabe andar reto". De acordo com a metodologia Logo, deve-se procurar caminhos que levem a solução e nunca "dar de graça", pois neste caso ela não será vivenciada e consequentemente não será assimilada. O que na maioria das vezes é feito, é procurar meios para chegar ao conceito do "polígono explodido" descrito por:

```
repita 360 [pf 1 pd 1]
```

Isto é uma circunferência utilizando o conceito que advém da idéia de limite, do cálculo diferencial, de que circunferência é um polígono cujo comprimento de cada lado tende a zero. Portanto conceitos sofisticados e complicados são explorados, e isto ocorre, por exemplo, do desejo de desenhar a roda de um caminhão [Valente, 1989].

Esta parte de exploração do ambiente através da execução de comandos no modo direto permite uma familiarização com a forma de escrever comandos dentro da linguagem e com o ambiente computacional de modo geral. Depois da construção de formas geométricas elementares o usuário começa a querer combinar formas na construção de desenhos mais elaborados e então advem a necessidade de introduzir o conceito de procedimentos.

A metáfora utilizada ao introduzir a definição de procedimentos é a de "ensinar" novas palavras à Tartaruga, que serão tratadas como as previamente definidas.

Computacionalmente e em estratégias de resolução de problemas, a idéia de procedimento é muito poderosa. Dentro da filosofia Logo, com a possibilidade de definir procedimentos se está

imitando a capacidade que as pessoas tem quando aprendem algo. Quando uma pessoa domina uma habilidade ou conceito, não precisa mais pensar sobre ele, ele é simplesmente utilizado em tarefas mais complicadas. Esta é mais uma razão que justifica começar a explorar a linguagem sob seu paradigma procedural.

O conceito de variável em Logo é introduzido através do uso de parâmetros em procedimentos e na maioria das vezes fica restrito a este contexto. A idéia é a de generalizar uma determinada solução de forma a que ela não precise ser reescrita para produzir, por exemplo, um mesmo desenho, só que com outras dimensões.

Recursão é um dos conceitos mais difíceis de ser assimilado, principalmente quando há a necessidade de explorar seu aspecto mais obscuro que é o retorno. Recursão é introduzida quando se deseja fazer desenhos, onde não se tem "a priori" o número de repetições que serão efetuadas de uma dada forma básica. Inicialmente, portanto, recursão é introduzida como uma forma de fazer repetição condicional. Em um segundo passo, apresenta-se o retorno e como fazer desenhos que utilizem operações no retorno, como os fractais. Este passo é muito difícil de ser compreendido,

embora os resultados obtidos sejam satisfatórios. Programadores iniciantes chegam a explorar formas gráficas recursivas bastante complexas e inclusive ser capaz de produzir o desenho de fractais.

Entretanto a passagem do modo gráfico para o modo de processamento simbólico é bastante traumática. Altera-se radicalmente a natureza dos problemas tratados e fundamentalmente, altera-se a forma de representar soluções, pois muda-se o paradigma de programação que passa a ser funcional. Além disso não se tem mais a Tartaruga que tem o papel de "andar" pelo programa, servindo de um importante meio para compreender o comportamento do programa e do computador.

Como em toda abordagem da linguagem, o processo de mudança tem início em uma fase exploratória do novo contexto. A partir disso, passa-se a construir operações básicas de percurso em listas, procura de elementos, construção de estruturas, retirada de elementos, inserção de elementos, etc. A idéia é a de procurar instrumentar o aluno, de forma a que ele tenha ferramentas para pensar sobre a solução de problemas mais complexos.

Os problemas principais de conceitos de programação aparecem nesta fase. A seguir faço uma descrição mais profunda destes problemas, procurando situar em que momento do aprendizado eles começam a ocorrer.

4.4 Problemas Conceituais

4.4.1 Procedimentos e Fluxo de Execução

Para utilizar em programação a estratégia de quebrar um problema em partes, é essencial o conceito de procedimentos e sub-procedimentos. Segundo [Valente,1987] este é um conceito difícil de ser assimilado, devido a dificuldade de entendimento do processo dinâmico ou fluxo de execução. As pessoas se fixam em estados estáticos, no resultado final de uma série de instruções, e não nas transformações que ocorrem durante a execução destas instruções.

Quando se está trabalhando com os comandos primitivos de Logo que controlam a Tartaruga, em modo direto, existe uma

correspondência um-a-um: comando-resultado. Já quando os comandos são agrupados, o resultado do grupo de comandos é, muitas vezes, muito mais do que é esperado dos comandos individuais. O comando REPITA para desenhar circunferência, visto anteriormente, é um exemplo clássico deste fenômeno. Através dele se tem o andar girando, que a princípio parece impossível com a Tartaruga. Ao escrever procedimentos é preciso pensar na síntese, ao invés dos comandos isolados. A esta dificuldade de entendimento deve-se estar atento quando da introdução do conceito de subprocedimentos. A analogia de que é mais uma palavra que está sendo ensinada não é suficiente.

Outro problema é com relação ao entendimento do fluxo de execução. Por exemplo, ao ser apresentado o procedimento:

```
ap casa          ap paredes       ap telhado
paredes          .....            ......
janela           fim              fim
telhado
chaminé          ap janela        ap chaminé
fim              .....            ........
                 fim              fim
```

A maioria das pessoas, que já programavam Logo há algum tempo, às quais solicitei que dissessem qual o fluxo de execução disseram que seria:

casa ---> paredes ---> janela ---> telhado ---> chaminé

ao invés de,

casa ---> paredes ---> casa ---> janela ---> casa ---> .....

Como o resultado final na maioria das vezes é equivalente, adquirir este modelo de como o procedimento casa será executado não acarreta grandes problemas. Mas entender como exatamente a chamada de procedimentos funciona é essencial para o entendimento de recursão, por exemplo. A visibilidade da forma como é feita a execução dos subprocedimentos não é provida pelo ambiente Logo.

4.4.2 Variáveis

O conceito de variável em Logo é introduzido através do uso de parâmetros em procedimentos e quase sempre fica restrito a este contexto. Como consequência observa-se que fora deste contexto há

uma grande dificuldade em utilizar este conceito essencial em programação, principalmente sob o paradigma procedural. Por exemplo, a flexibilidade em desenhar vários quadrados de tamanhos diferentes através de comandos isolados, é obtida usando um parâmetro em procedimento que especifique o tamanho do lado desejado. Esta idéia é facilmente dominada e utilizada em diferentes desenhos com um número variado de parâmetros. Entretanto mesmo utilizando variáveis largamente como parâmetro, nota-se a não assimilação do conceito ao observar programas como o que segue:

```
aprenda quadrados
quadrado 16
quadrado 20
quadrado 24
quadrado 28
quadrado 32
quadrado 36
quadrado 40
fim
```

Normalmente este procedimento também é escrito da seguinte forma:

```
aprenda quadrados :x
quadrado :x + 4
fim
```

Com isto obtem-se o resultado desejado de forma mais compacta, mas ainda persiste o desentendimento sobre o que se passa com o valor da variável (o programador não altera "fisicamente" o valor da variável, mas deixa para o programa fazê-lo).

Este procedimento poderia ser escrito como:

```
aprenda quadrados
local "lado
coloque 16 "lado
repita 7 [ quadrado :lado  coloque :lado + 4  "lado]
fim
```

Note que para obter essa solução é preciso ter entendido o conceito de variável em seu sentido mais amplo e abstrato, e

notadamente isto não é obtido da forma como o conceito é apresentado.

É preciso desenvolver procedimentos como o apresentado acima para que este conceito seja assimilado, e trabalhar de forma convencional, desenvolvendo formas de teste de mesa, onde sejam ressaltadas as variações nos valores de variáveis. Novamente deve-se encontrar formas de enriquecer o ambiente para facilitar o entendimento.

4.4.3 Recursão

Recursão é um dos conceitos mais difíceis de ser assimilado, principalmente quando há a necessidade de explorar seu aspecto mais dependende de como o procedimento será efetivamente executado, que é o retorno. Mais uma vez aparece a dificuldade de lidar com coisas acontecendo automaticamente, "invisivelmente" dentro do computador.

Ao trabalhar com a parte gráfica não se percebe claramente estas deficiências de entendimento. Isto talvez seja devido a um certo empiricismo que envolve a solução de problemas gráficos,

aliado à facilidade que existe em acompanhar a solução e consequentemente fazer correções que conduzam à solução desejada, mesmo que sejam correções tipicamente "remendos" e que não exigem o entendimento do conceito.

Dentro da exploração gráfica obtem-se procedimentos extremamente sofisticados como:

```
aprenda fractal :comprimento :fragm
se :fragm = 0 [ pf :comprimento pare ]
fractal :comprimento/3 :fragm-1
pe 60
fractal :comprimento/3 :fragm-1
pd 120
fractal :comprimento/3 :fragm-1
pe 60
fractal :comprimento/3 :fragm-1
fim
```

O que é esperado é que uma pessoa que escreve este tipo de procedimento domine a técnica de recursão.[Give'on, 1989] Entretanto na realidade isto não ocorre. Através de um processo

de tentativa e erro, que o ambiente gráfico facilita, a pessoa adquire padrões de como construir figuras deste tipo, altamente recursivas, identificando na forma final desejada o que denomina de "foco de recursão" e colocando em correspondência dentro do procedimento a respectiva chamada recursiva.

Inclusive a identificação de "focos de recursão" tem sido explorada como técnica de ensino de recursão, apresentando programas padrões para desenhar fractais, ou árvores, ou percursos de estruturas, etc. [Harvey, 1985].

Outro fato que clarifica o não entendimento é que foi verificado que mesmo pessoas que escrevem este tipo de procedimento não tem ainda bem claro a idéia de como é o fluxo de execução de instruções dentro de um procedimento.

Todas estas falhas de entendimento são mais claramente observáveis quando da passagem da parte gráfica para a parte de manipulação simbólica. Um exemplo é o apresentado em [Valente, 1987], onde era desejado escrever números de 1 a 3 em ordem decrescente e em seguida em ordem crescente e foi apresentado o seguinte procedimento:

```
aprenda números :n
se :n=1 [ pare ]
números :n-1
escreva :n+1
escreva :n+2
escreva :n+3
fim
```

Percebe-se a tentativa de controlar concretamente um processo automático, mostrando o não domínio da noção da volta da recursão ignorando a pilha de recursão. Nota-se a existência de um modelo de que recursão é repetição, mas um procedimento sempre é executado até o comando Fim.

Com este tipo de dificuldade, não se consegue avançar tranquilamente da parte gráfica para a simbólica onde problemas básicos como o de procurar um determinado elemento em uma lista, contar o número de elementos de uma lista, etc., envolvem necessariamente a utilização de conceitos como variável, fluxo de execução e recursão, que ficam fragilmente formalizados dentro da parte gráfica. Isto leva a maioria dos usuários de Logo a só utilizarem a parte gráfica, o que conduz erroneamente a se

identificar Logo somente com atividades de fazer desenhos. Existe uma grande dificuldade de transpor os conceitos de programação utilizados em atividades gráficas para atividades que manipulem números, estruturas de dados como listas, sentenças , palavras, etc.

4.5 Conclusão

Mesmo sendo Logo uma linguagem que objetiva facilitar o aprendizado de programação, existe uma barreira na transposição da parte gráfica para a simbólica. Como facilitar esta passagem e consequentemente facilitar o aprendizado de conceitos essenciais de programação, respeitando o aspecto metodológico básico do Logo que é o de proporcionar um ambiente de aprendizagem onde o conhecimento não é passado e sim desenvolvido através da interação com os objetos do ambiente, é o que norteou meu trabalho de pesquisa.

Tentando localizar as fontes de problemas, o que procurei fazer foi fortalecer o Logo de forma a que a mudança de paradigma pudesse ser efetuada de forma mais suave, tendo-se um ambiente menos árido de desenvolvimento de operações, com uma maior

visibilidade da execução que é tão essencial para o entendimento dos conceitos básicos de programação.

Todo o desenvolvimento do trabalho foi baseado em resultados de pesquisas sobre a importância de modelos mentais no entendimento de domínios de conhecimento. Assumi que também em programação as pessoas são guiadas por modelos mentais do funcionamento da linguagem, que são construídos gradativamente a partir da interação com o ambiente de programação. A partir daí deve-se ter no ambiente ferramentas ou interfaces que auxiliem na construção de modelos adequados. No próximo capítulo faço um maior detalhamento destes aspectos teóricos.

Capítulo 5

Modelos Mentais

5.1 Introdução

As pessoas quando interagem com o ambiente, com outras pessoas e com artefatos da tecnologia elas formam um modelo mental de si próprias, das pessoas e das coisas com as quais estão interagindo.

Estes modelos provêem poder de entendimento, capacitando as pessoas a predizer e explicar aspectos da interação. Ao mesmo tempo estes modelos são modificados pela própria interação. A importância de modelos mentais na aquisição de habilidades humanas é atualmente um tópico de pesquisa bem representado em Ciência Cognitiva e Inteligência Artificial.

Da Ciência Cognitiva advem todo um conjunto de técnicas para investigar o que se passa na mente humana, e atualmente é possível ser bastante ambicioso sobre testar complexas teorias sobre como as pessoas entendem informação natural.

A Inteligência Artificial por sua vez vem desenvolvendo poderosos formalismos através dos quais pode-se explicitamente

denotar teorias sobre a representação do conhecimento e processamento humanos. Na última década, modelos computacionais progrediram dos primeiros modelos que enfatizavam o fluxo de informação para um requintado formalismo para representar tanto dados como processos, dentro de uma estrutura uniforme. Há três décadas atrás, Piaget tinha apenas modelos matemáticos para usar como formalismo para representação do conhecimento. Ele modelou, por exemplo, o tipo de processos de pensamento que são adquiridos na adolescência através de 16 relacionamentos algébricos booleanos. Este tipo de representação é uma limitação, pois não nos permite captar mais que a complexidade da representação do conhecimento humano.

Na confluência destas duas áreas estuda-se modelos mentais, desenvolvendo-se pesquisas que buscam caracterizar, através de um exame cuidadoso, o modo de entendimento das pessoas nos mais variados domínios de conhecimento, como circuitos elétricos [Kleer, 1983], cálculo matemático [Bundy, 1983], física elementar [Larkin, 1983], temperatura e calor [Wiser, 1983], etc.

Do ponto de vista psicológico pode-se embasar a conceituação de modelos mentais na teoria cognitiva de Piaget, que apresenta

a idéia análoga de esquema mental. Para Piaget esquema mental representa aquilo que pode ser repetido e generalizado (modelo) numa ação. Por exemplo, existe o esquema mental do ato de empurrar um objeto, não interessando se o objeto está sendo empurrado com a mão, com uma vassoura, etc. [Piaget, 1970]

Os resultados destas pesquisas vem sendo utilizados na construção de sistemas especialistas, na concepção de sistemas computacionais realmente amigáveis, no desenvolvimento de material instrucional, manuais, etc.

Também em programação, principiantes são guiados por modelos mentais de como o código do programa controla as operações do computador. Mas o computador, como sistema físico é extremamente obscuro e fornece um "feedback" muito pobre no sentido de auxiliar um principiante a entender seu funcionamento e consequentemente formar um modelo mental adequado, que o possibilite interagir corretamente. Para exemplificar, se em Logo escrevo um procedimento que inverte uma palavra de entrada, o que obtenho como resposta pode ser a palavra invertida, ou então a palavra sem alguns caracteres, ou a palavra na mesma forma que entrou, etc. Sobre o processo, nada é mostrado. Daí meu interesse

em fornecer meios de visualização de um processo computacional, de forma a auxiliar a formação deste modelo mental.

Neste capítulo procurarei caracterizar modelos mentais através da análise de algumas pesquisas desenvolvidas nesta área, procurando ressaltar resultados teóricos que foram importantes no desenvolvimento de minha pesquisa.

5.2 Caracterização de Modelos Mentais

Norman em [Norman, 1983] ressalta que ao se analisar modelos mentais, com relação a sistemas físicos, é necessário considerar quatro componentes:

.o sistema objetivo, que por definição é o sistema que a pessoa está aprendendo ou usando.

.o modelo conceitual do sistema objetivo, que são ferramentas para auxiliar o entendimento ou ensino de um sistema. São inventados por professores, designers e engenheiros com o objetivo de prover uma representação adequada do sistema

objetivo. É desejável que sejam precisos, consistentes e completos.

.o modelo mental do usuário sobre o sistema objetivo, é o que as pessoas realmente tem em suas cabeças e que guia o seu uso do sistema.

.a concepção do cientista sobre o modelo mental, que é, claramente, um modelo do modelo.

Modelos mentais são naturalmente evolutivos. Através da interação com o sistema objetivo, pessoas formulam modelos mentais deste sistema. Estes modelos não necessitam ser tecnicamente precisos e usualmente não são, mas precisam ser funcionais. Uma pessoa através da interação com o sistema, continuamente modifica o modelo mental, no sentido de obter um resultado trabalhável, ou seja, que funcione. Modelos mentais são limitados por fatores como o embasamento técnico do usuário e fundamentalmente, pela experiência prévia com sistemas similares.

O propósito de um modelo mental é permitir a pessoa entender e antecipar o comportamento de um sistema. Isto significa que o

modelo mental precisa ter um poder predectivo, ou seja, deve ser possível para uma pessoa "executar" seus modelos mentalmente [Williams, 1983; Norman, 1983; Kleer, 1983]. No contexto de programação, executar o modelo mentalmente é equivalente a predizer o resultado de um procedimento sem ter que executá-lo no computador.

Idealmente deveria haver uma relação direta e simples entre o modelo conceitual e o modelo mental. Mas isto nem sempre é conseguido e o que se procura é uma correspondência entre os parâmetros e estados do modelo conceitual e o que ele procura descrever. Com isso se auxilia o usuário do sistema na formação de um modelo mental adequado.

Williams em [Williams, 1983] ressalta três características básicas de modelos mentais: são compostos de objetos autônomos com uma topologia associada , são executáveis e podem ser decompostos.

Central a esta concepção de modelos mentais é a noção de objeto autônomo. Um objeto autônomo é um objeto mental com uma explícita representação de estado, uma explícita representação de

suas conecções topológicas com outros objetos e possui um conjunto de parâmetros internos. Associado com cada objeto autônomo existe um conjunto de regras as quais modificam seus parâmetros e especificam seu comportamento.

Desde que um modelo mental é uma coleção de objetos autônomos conectados, executar um modelo mental corresponde a modificar os parâmetros do modelo através de uma propagação de informação, usando regras internas e a topologia específica.

Altera-se um modelo mental quando objetos autônomos mudam de estado ou novos objetos são acrescentados. A mudança de estado é distinta da mudança do valor corrente dos parâmetros de um objeto. A mudança de estado consiste da substituição de um conjunto de regras de comportamento por outro.

Outra questão que surge quando se fala sobre modelos mentais é **"Por que as pessoas usam modelos mentais?"**

Modelos mentais auxiliam o raciocínio humano de diversas maneiras. Eles são utilizados como uma máquina de inferência para predizer o comportamento de um sistema [Williams, 1983]. Também

podem ser utilizados para produzir explicações e justificativas [Kleer, 1983; diSessa, 1983; Gentner, 1983; Wiser, 1983]. Adicionalmente eles servem como um dispositivo mnemônico para facilitar relembrar [Williams, 1983]. Mas o ponto central, que todos os autores enfatizam, é o uso de modelos mentais na produção de explanações e justificativas.

5.3 Natureza dos Modelos Mentais

Young em [Young, 1983] compara diferentes modelos de calculadoras de mão, e usa análise de protocolo para suportar seus argumentos teóricos de que diferentes modelos levam a diferentes tipos de performance. Ele afirma que é amplamente aceito que a habilidade das pessoas em usar um sistema interativo depende dela ter um acesso a uma espécie de modelo mental, que pode ser visto como sendo uma representação ou metáfora que o usuário adota para guiar suas ações e interpretar o comportamento do sistema. Ele ressalta diferentes formas da natureza destes modelos (M) sobre um sistema interativo (S):

.**Modelo de Analogia Forte ("Strong Analogy Model")**: S é suficientemente familiar a outro sistema S' de forma que a

representação do sistema S' serve como um modelo M para S. Um exemplo, é quando se afirma que "uma unidade de vídeo de computador é como uma máquina de escrever", e portanto sabendo como datilografar dá informação sobre como utilizar a unidade de vídeo. Deve-se observar que isto não pode ser visto como um modelo final, pois não informa nada sobre como o sistema S' é representado, e em algum ponto a cadeia de analogias deve ser quebrada e outro modelo adotado. Inclusive pode-se mencionar com relação a esta analogia entre vídeo e máquina de escrever, que ela ocasiona problemas sérios de interação com o computador. Um exemplo é o mencionado em [Valente, 1987] onde observou-se que principiantes quando querem "apagar" algo que fizeram errado, simplesmente desligam o monitor de vídeo.

.**Modelo Substituto** ("Surrogate Model"): M é fisicamente ou notacionalmente análogo ao mecanismo de S, e pode ser usado para fornecer respostas sobre o comportamento de S. É essencialmente uma descrição "mecanicista" de como o sistema funciona. Um exemplo, são os tradicionais fluxogramas ou diagramas de blocos, utilizados em programação para representar através de formas gráficas como é o fluxo de execução de um programa. Outro exemplo que pode ser mencionado, são as cartas sintáticas de uma

linguagem de programação, que especificam como devem ser escritas as construções de uma linguagem, através de uma série de esquemas gráficos. As cartas também mostram como o analisador sintático de uma linguagem funciona.

.Modelo do Vocabulário ("Vocabulary Model"): M é um conjunto de termos dentro do qual o conhecimento sobre S é codificado. Este modelo focaliza atenção sobre os termos mentais dentro dos quais o usuário codifica informação sobre o sistema. Esta proposta sugere que é mais fácil aprender certos fatos que podem ser expressos dentro de um vocabulário existente, que outros que envolvem a aquisição de novos termos. Este modelo é proposto por Newell e deriva diretamente dos protocolos verbais coletados quando da análise de sujeitos resolvendo problemas [Newell, 1972]. Este modelo é semelhante ao modelo de analogia e com os mesmos problemas. Neste modelo uma possível fonte de erro durante o aprendizado é a tentativa de codificar uma nova informação dentro de um vocabulário existente, mas que não corresponde a natureza da informação. Para exemplificar, tem-se problemas de uso do Logo, por usuários de linguagens do tipo do Pascal, quando do uso de variáveis globais. Isto ocorre porque não se pode

adotar o mesmo conceito, embora a terminologia seja a mesma, dado o escopo dinâmico da linguagem Logo.

Na definição de um modelo conceitual, sobre um sistema ou processo deve-se levar em conta qual a natureza de modelo mental que se está considerando, para que se obtenha a desejável correlação entre o modelo mental e o modelo conceitual. O tipo de modelo que fundamentou meu trabalho, como um esquema de representação que visa auxiliar o usuário a adquirir conhecimento, e consequentemente adquirir um modelo adequado, do funcionamento de um procedimento ou programa, foi o modelo substituto que discuto em mais detalhe na próxima seção deste capítulo.

5.4 Modelo Substituto ("Surrogate Model")

Um modelo substituto M para um sistema S é essencialmente uma noção familiar de um modelo de trabalho o qual explica como S funciona como um mecanismo. Tipicamente M apresenta uma abordagem bastante simplificada de S, de forma a ter "mais o sabor de uma estória de ficção do que de uma elaborada descrição de engenharia" [Young, 1983].

M pode ser um modelo físico, ou seja um análogo, como um aeromodelo, mas mais comumente ele existe apenas no papel, escrito em alguma notação formal (como a matemática) ou informal (representações gráficas).

M é chamado de substituto porque ele pode ser usado no lugar de S para responder questões sobre o comportamento de S. Em adição a ser amplamente utilizado em Ciências e Engenharia, modelos substitutos tem sido utilizados como um meio de fornecer a programadores novatos um esquema de como funciona o interpretador de uma linguagem de programação [duBoulay, 1981; Mayer, 1981].

Modelos substitutos são guiados pela propriedade predectiva dos modelos mentais e, primariamente, podem ser vistos como um simulador do sistema objetivo. Também são de grande auxílio, ao prover uma estrutura para o funcionamento do sistema e como consequência auxiliar a pensar sobre ele.

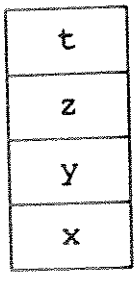
Para exemplificar, vou relacionar dois modelos substitutos descritos em [Young, 1983], juntamente com as limitações que tais modelos apresentam.

Exemplo1: Calculadora de Notação Pós-fixa (RPN)

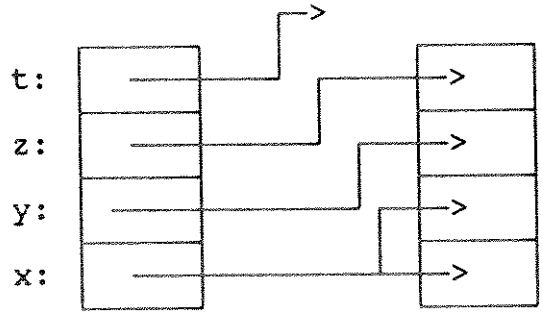
Certos projetos de calculadoras empregam o cálculo pós-fixo, ou seja, os operadores aritméticos são escritos depois dos operandos. Portanto para expressar a soma "2+3" uma RPN escreve "23+".

A apresentação padrão de uma RPN é feita através de um modelo conceitual substituto como o de pilha, descrito a seguir.

(a)

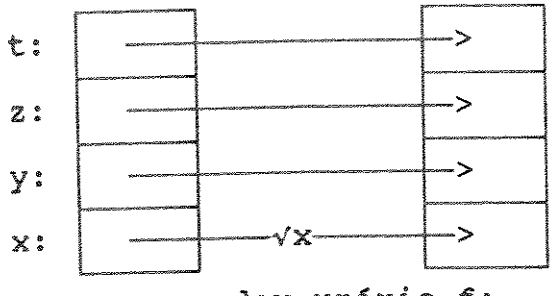


(b)



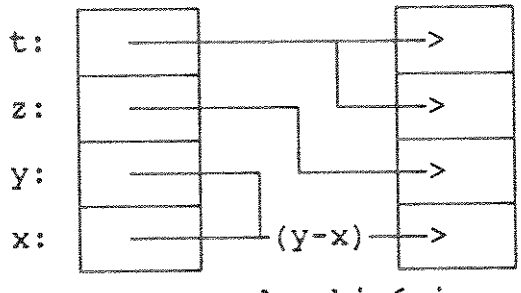
ENTRE:
t: = z
z: = y
y: = x

(c)



operador unário f:
x: = f(x)

(d)



operador binário g
x: = g(x,y)
y: = z
z: = t

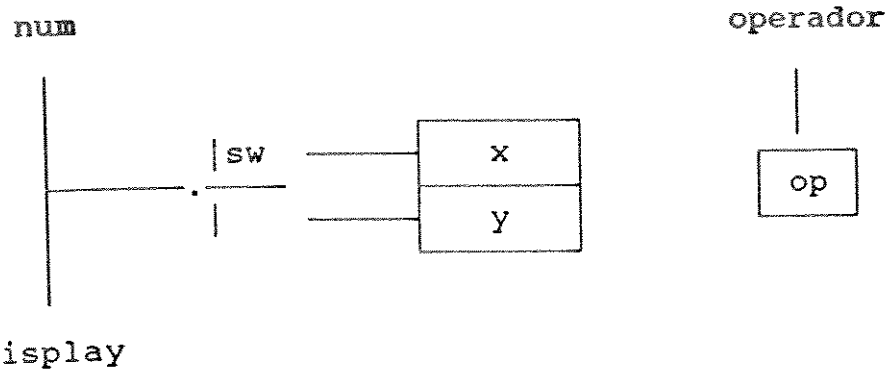
Como pode ser observado, o modelo consiste de uma pilha de quatro registradores, cada um com a capacidade para armazenar um número. Os registradores são chamados x,y,z e t, e é sempre o número que está em x que é mostrado no display. A pilha é utilizada para especificar o efeito de várias operações.

Esta pilha provê uma elegante ilustração do que um modelo substituto pode ser. Poucos detalhes são mostrados e o modelo de pilha é adequado para mostrar o comportamento da calculadora.

Exemplo2: Uma calculadora simples de 4 funções (FF)

Este segundo exemplo, o de uma calculadora bastante simples, algébrica no sentido de utilizar operação infixa, mas que não possui nenhuma definição de hierarquia de operadores. Além das teclas de números existe apenas a tecla '=' e as teclas das quatro operações.

O modelo substituto para esta FF é mostrado a seguir.




```
inicialização:      x:= y:= 0
                    op:= "+"
                    sw --> x        ; i.é., chave aponta para x

entra um num  n:    @sw := n        ; i.é., armazena n no
                                      registrador que sw aponta

pressiona a operação f: se for "=" então fazer a sequência
                        para o "="
                        op := f
                        y := x
                        sw --> y

pressiona o "=" :       x := op(x,y)
                        sw --> x
```

O modelo tem dois registradores internos, chamados x e y, e um apontador (sw) que indica qual número deverá ser mostrado e para qual registrador irá o número teclado. O diagrama também dá regras para o funcionamento desta máquina notacional.

Da mesma forma que o modelo substituto para a RPN este modelo prove uma completa descrição da calculadora e preve sua resposta para qualquer sequência de entradas.

Young afirma que, mesmo sendo estes dois modelos de calculadoras, fiéis às máquinas que representam, eles apresentam sérias restrições. Ele menciona três objeções, que de alguma forma podem ser ampliadas para qualquer modelo substituto deste tipo, que relaciono a seguir.

A primeira objeção diz respeito a sua limitada aplicação. Para tarefas que requerem solução de problemas, o substituto pode talvez ser útil como uma representação mental sobre a qual a solução do problema é baseada. Mas para tarefas mais sofisticadas o substituto parece praticamente irrelevante. O principiante, que nunca utilizou uma máquina de calcular antes, pode executar seus primeiros cálculos seguindo os princípios básicos mostrados no modelo. Mas rapidamente ele aprende algumas sequências familiares que cortam caminho na solução de um problema. Estas sequências estão muito adiante do que é mostrado no modelo.

A segunda objeção é que o modelo substituto falha em captar diversas características salientes de uma calculadora. Pode ser observado que ambos os modelos tratam suas entradas como uma desestruturada sequência de números, operadores e sinais de igualdade (=s). Um problema específico ao modelo da calculadora FF pode ser salientado. Se for teclada a sequência "2+3==" se obtem a resposta 8 e se for pressionada a sequência "2+3=+=" se obtem 10. O modelo substituto entretanto dá a mesma resposta para ambos os casos. Portanto não existe maneira de inspecionar o modelo de modo a superar esta dificuldade e observar que a entrada de um operador interage com o valor "default" usado como segundo argumento. Portanto o modelo apesar de parecer tão claro tem problemas de design.

A terceira falha do modelo substituto deste tipo pode ser ilustrada imaginando-se um modelo substituto para uma máquina que tivesse teclas para fazer parentização e que também adotasse uma prioridade de operações. Seguindo o caminho destes modelos teríamos um modelo extremamente complexo, contendo uma série de pilhas , registradores, marcadores de prioridade, etc.

Segundo Young, todas estas falhas apontadas residem no fato de que um modelo substituto por sua natureza está centrado no dispositivo ou sistema, não levando em conta a tarefa do usuário. A sugestão é utilizar um modelo que dê mais atenção ao relacionamento entre a tarefa do usuário e o funcionamento do sistema por si só.

Mas por outro lado Young não considera que os modelos apresentados são estáticos e as falhas que ele denota são decorrentes de um processo dinâmico de interação. Se o modelo fosse construído de forma a possibilitar este nível de interação todos os problemas e falhas que ele menciona deixariam de existir. Isto é o que deve ser feito ao utilizar o computador como meio de fornecer modelos conceituais substitutos de sistemas interativos, como é o caso do funcionamento do interpretador de uma linguagem de programação.

5.5 Modelos Mentais e Conceituais em Programação

Antes de analisar a importância de modelos mentais e conceituais em programação, gostaria de deixar claro meu ponto de partida sobre o aprendizado de programação e o implícito

conhecimento do computador enquanto máquina.

Para principiantes em programação o computador não existe a nível de sistemas binários, portas, processador central e outras unidades básicas. O computador vai ser uma máquina definida em função das construções da linguagem de programação que estão aprendendo. Neste sentido, podemos ser extremistas e afirmar que se o principiante inicia programação através da linguagem Logo, o computador para ele vai ser uma máquina Logo, ou seja, que funciona de acordo com as instruções disponíveis na linguagem Logo, o mesmo irá acontecer com Pascal, ou qualquer outra linguagem de programação utilizada.

Daí definir-se que simplicidade e visibilidade são duas características importantes de uma linguagem de programação para principiantes, pois geralmente eles começam programação com nenhuma idéia sobre as propriedades da máquina notacional definida pela linguagem que estão aprendendo. A máquina notacional é o modelo idealizado do computador, definido através das construções da linguagem de programação, sendo portanto um computador conceitual idealizado. Para o principiante as

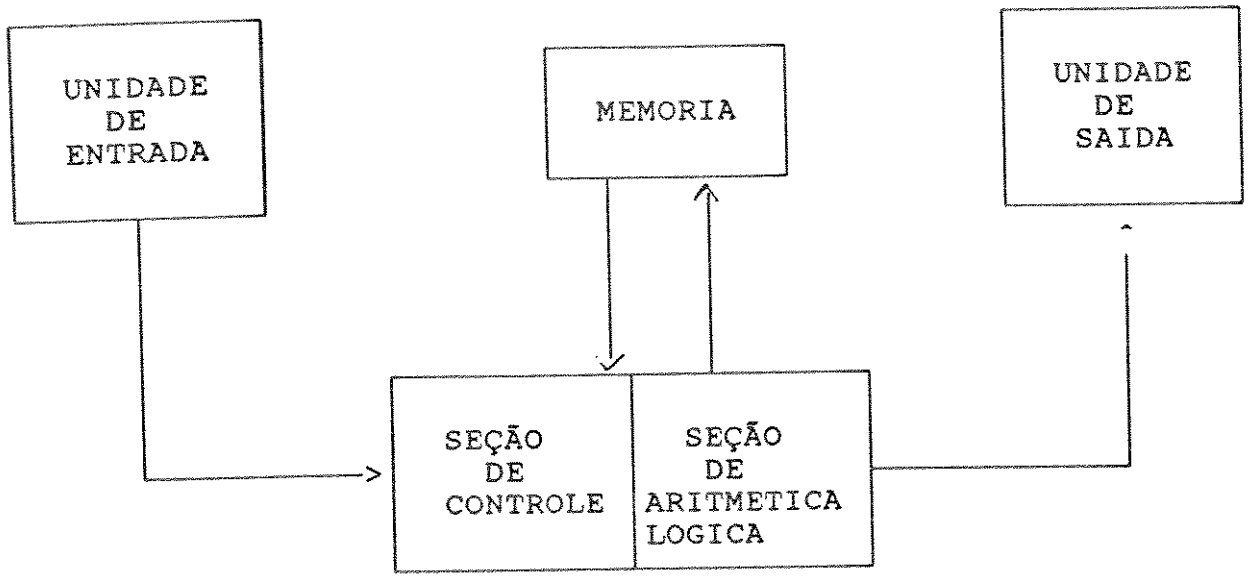
propriedades desta máquina são dependentes da linguagem e não do "hardware".

Esta simplicidade e visibilidade facilitam a construção de um modelo mental do computador e da sua forma de operação, este modelo é que irá guiar toda a interação do usuário a nível de programação.

O computador é uma máquina singular, opaca e extremamente complexa.

No ensino de programação sempre estão presentes tentativas de fornecer modelos do computador e de suas unidades básicas. Vou mencionar alguns exemplos de analogias e tentativas de construção de modelo, para ressaltar ao mesmo tempo a necessidade que é sentida em fazer com que o aluno crie um modelo e a dificuldade em encontrar modelos conceituais satisfatórios.

O primeiro modelo de computador que geralmente é apresentado pode ser visto a seguir:



Isto em absoluto pode-se dizer, pelos critérios descritos anteriormente, que seja um modelo adequado. Mostra de forma bastante superficial as "partes" principais do computador e seus relacionamentos primários. Mas não se consegue através dele explicar como a máquina opera um comando simples de saída, como "escreva 7+5".

Algumas analogias são utilizadas para tentar dar um modelo funcional das partes do computador, como por exemplo, dizer: "A seção de controle de um computador desempenha o mesmo papel do regente de uma orquestra. Assim como o regente irá dirigir a

orquestra de acordo com uma partitura musical que foi escrita pelo compositor, a unidade de controle irá comandar e dirigir todos os trabalhos do computador" [Baranauskas, 1986, pg. 64].

Muito trabalho é feito no sentido de tentar fazer com que os alunos entendam a memória, dada sua importância no conceito de variável. De princípio, a definição formal da memória como sendo uma coleção de pequenos circuítos, chamados FLIP-FLOP em um CHIP que podem estar ou não conduzindo corrente, é absolutamente sem significado para o iniciante. Comumente é utilizada então uma analogia, fazendo os alunos pensarem na "memória do computador como uma grande caixa de correio dividida em pequenas caixas postais numeradas. Cada caixa irá possuir um número que a identifica. Também na memória do computador, cada posição é identificada por um número que chamamos de ENDEREÇO. Quando precisarmos utilizar uma posição de memória em vez de utilizar o seu endereço, daremos um nome a ela e a isto chamaremos de VARIAVEL" [Baranauskas, 1986, pg.85].

As deficiências desta analogia são evidentes. Em uma caixa do correio podemos colocar diversas correspondências, e no caso de uma posição de memória, podemos ter acesso somente a uma

informação por vez. Quando os alunos refazem seu modelo de caixa de correio, para uma caixa de correio especial que representa a memória, são definidas as variáveis estruturadas, a confusão retorna e a analogia se perde. Como um mesmo nome pode representar diversas caixas e muitos valores diferentes?

O que se observa é que os alunos que são conduzidos por este caminho de analogias sentem muita dificuldade em trabalhar com variáveis estruturadas do tipo matrizes ou listas. Certamente eles não tem meios de modificar um modelo bem instalado de modo a acomodar este novo objeto, ou então, de alterar as regras e parâmetros do objeto variável.

Como já foi discutido no Capítulo 2, também na introdução da idéia do que é um programa, ou um algoritmo, é comum tentar fazer analogia com algoritmos de atividades corriqueiras e dirigidas à pessoas. E isto, como já foi analisado no Capítulo 2, também não se mostra adequado.

Uma das conclusões que se pode extrair é, que apesar de analogia ser um importante recurso que as pessoas utilizam para ajudá-las a estruturar domínios não familiares [Gentner, 1983],

no domínio de programação não se mostra efetiva. Portanto um modelo conceitual neste domínio, não pode ser baseado em analogias.

Modelos mais abstratos e próximos da idéia de modelos substitutos são também amplamente utilizados e tem se mostrado mais eficazes. Muitos deles estão discutidos no Capitulo 2, como os diagramas de execução, árvores de ativação, etc. Estes modelos de forma geral objetivam mostrar o funcionamento do computador em função das construções da linguagem e não do comportamento a nível de "hardware", ou seja, o sistema objetivo é a máquina notacional definida pela linguagem. Acredito, que como auxílio a programação, este é o modelo que deve ser explorado. Consequentemente, recursos utilizados no sentido de ajudar o usuário a visualizar a operação do programa são muito úteis no sentido de facilitar à atividade de programação.

Partindo desta análise advoga-se que instrução deve explicitamente ajudar o estudante a construir um modelo mental de como o computador funciona, pois se o funcionamento da linguagem é um mistério, o estudante não será capaz de escrever programas.

Para deixar mais clara a idéia de que também em programação as pessoas são guiadas por modelos mentais, vou me referir a alguns aspectos do processo de aprender programação em Logo, alguns deles já abordados no Capitulo 4.

O subconjunto gráfico da linguagem Logo que controla a Tartaruga é um exemplo de ambiente facilitador para o aprendizado de programação, dado que possui as características desejáveis de simplicidade e visibilidade. Com uma pequena sequência de instruções obtem-se programas com efeitos gratificantes. Um conjunto de mais ou menos dez instruções e mais a estrutura de procedimentos e sub-procedimentos torna possível criar um rico e interessante, mas claramente limitado, conjunto de ações da máquina.

Este subconjunto gráfico é tradicionalmente a "porta de entrada" do Logo. Através da execução de programas que fazem desenhos, de alguma forma se tem espelhado o comportamento passo-a-passo do programa e consequentemente torna-se mais claro o funcionamento da máquina notacional, visualizando o funcionamento de cada construção da linguagem.

Dentro deste ambiente existem diversas componentes que facilitam o usuário a formar um modelo mental do funcionamento do computador.

O computador é visto como um traçador de gráficos que se comporta de forma semelhante a como uma pessoa traça gráficos. Muitas vezes, é solicitado que a pessoa se coloque no lugar da Tartaruga e quase sempre é isto que ela faz quando depura seus procedimentos.

Muitos objetos (conjunto de instruções, Tartaruga, parâmetros) e comportamentos importantes são incorporados ao modelo mental que a pessoa está desenvolvendo sobre o funcionamento do computador, como a sequencialidade na execução de instruções, o procedimento produzindo um resultado visual na tela, parâmetros para generalizar soluções, etc.

O modelo da sequencialidade de execução de instruções guia as pessoas a descreverem o fluxo de execução de um procedimento do tipo:

ap casa	ap paredes	ap chaminé
paredes	.....	.....
telhado	fim	fim
chaminé		
janela	ap telhado	ap janela
fim	.....	.....
	fim	fim

como sendo, casa -->paredes --> telhado ---> chaminé ---> janela

Como o "feedback" que é dado pelo computador não explicita que este não é o fluxo, as pessoas persistem com esta idéia de fluxo e com este modelo de funcionamento.

Sem muita dificuldade é incorporado o comportamento iterativo, por ser uma idéia decorrente do sequencial e que somente simplifica a escrita de comandos. O mesmo não acontece com comandos condicionais. Dificilmente se observa principiantes escrevendo comandos condicionais com a estrutura se/então/senão, que implica em uma quebra maior da sequencialidade estrita, pois dependendo do resultado da expressão utilizada no comando, algumas instruções deverão ser "saltadas".

O uso do modelo iterativo, de repetição, guia o entendimento de procedimentos recursivos. Isto pode ser observado, quando escrevem procedimentos do seguinte tipo:

```
ap contar :n
esc :n
se :n = 0 [ pare ]
contar :n-1
esc :n+1
esc :n+2
esc :n+3
fim
```

para escrever a sequência de números 3 2 1 0 1 2 3.

Através deste exemplo, está claro que o aluno vê recursão como uma repetição. Também pode ser observada a necessidade em controlar o processo concretamente, que o aluno abstrai do uso inicial da linguagem no modo direto. E mais uma vez a resposta que o computador vai dar a este procedimento não irá auxiliá-lo a formular o correto modelo de recursão que é o de cópia, pois ele não consegue visualizar o processo de execução.

5.6 Conclusão

Estes problemas relatados, são agravados, no aprendizado de Logo, quando se tenta avançar da parte gráfica para a parte de manipulação simbólica. O modelo do computador não é mais o de um traçador de gráficos, precisa ser modificado sensivelmente.

Esta mudança é agravada pois Logo perde a característica de visibilidade. Não se pode mais acompanhar a execução do programa. O que é obtido é simplesmente o resultado, na forma de listas, sentenças, palavras ou números. Torna-se difícil entender como o computador chegou ao resultado, ou não, pois não existem mais maneiras diretas de acompanhar a execução do programa. Além disso, muda-se o paradigma de programação, ou seja, agora deve ser reformulado também o modelo de procedimento, acrescentando o comportamento funcional. De acordo com a teoria de Williams em [Williams, 1983] as regras de comportamento associadas ao objeto autonômo procedimento devem ser substituídas, de forma a acomodar a definição de procedimento que é uma operação, ou então, um novo objeto denominado operação deve ser incorporado ao modelo.

Portanto, a perda da visibilidade tão marcante na parte gráfica, traz sérias dificuldades no avanço em programação.

Foi com o objetivo de auxiliar a formação de um modelo mental adequado do funcionamento da linguagem, que implementei o sistema que descrevo nesta tese. Parti do princípio que dado que modelos mentais são por natureza evolutivos e gradativamente alterados de acordo com a interaçãao. Então, a observação do comportamento do código de um programa, de acordo com um modelo conceitual que enfatiza aspectos de fluxo de execução, chamada e retorno de procedimentos, retorno e escrita de valores, é um "feedback" extremamente útil para que as pessoas possam gradativamente adaptar e tranformar seus modelos.

Capítulo 6

Objetivo e Metodologia

6.1 Introdução

Neste capítulo descrevo o objetivo da pesquisa, procurando identificar o porque deste tópico de estudo, bem como dar as justificativas relevantes a escolha deste tema e do contexto que foi trabalhado.

Também é feita uma descrição da metodologia de trabalho utilizada. Não é feita propriamente a descrição rígida de uma metodologia de pesquisa. O que é relatado é o caminho seguido que conduziu a implementação do sistema, que descrevo no próximo capítulo, bem como a abordagem escolhida para fazer a observação de uso.

6.2 Objetivo

No aprendizado de programação existem muitos conceitos difíceis de serem assimilados, tais como, variáveis e seu escopo de validade, super e sub-procedimentos, fluxo de execução, recursão, iteração, etc.

Tem-se como hipótese, que uma das dificuldades básicas para entender tais conceitos é que não se consegue enxergar como realmente o computador executa um programa, dificultando a criação de um modelo mental adequado que guie a utilização destes conceitos em programação.

Dada a opacidade do computador enquanto máquina e a natureza abstrata dos conceitos computacionais, deve-se utilizar representações caso se deseje tornar de alguma forma "visível" o processo de execução. E como o processo é dinâmico deve-se ter representações dinâmicas que irão constituir um modelo conceitual interativo do funcionamento de programas.

Considerando-se a linguagem Logo aliada a sua metodologia de aprendizado o objetivo do trabalho foi ampliar o ambiente de programação, criando um sub-ambiente onde são apresentadas representações alternativas de um programa em execução, objetivando auxiliar o entendimento de conceitos computacionais.

Com este objetivo pode-se subdividir a pesquisa em três áreas de estudo:

a) estudo de como pessoas entendem conceitos computacionais

b) desenvolvimento de sistemas computacionais que forneçam representações alternativas da execução de um programa, em tempo de execução. .

c) análise experimental dos resultados obtidos do uso do sistema desenvolvido, de modo a poder verificar se efetivamente auxiliam a compreensão dos conceitos.

6.3 O que Originou a Escolha deste Tema

A pesquisa aqui descrita foi definida a partir da análise de quanto é essencial o uso de representações no entendimento de conceitos e processos abstratos e da observação da importância da representação de um problema quando se está trabalhando com solução de problemas. Aliado a estes dois ítens considerou-se as pesquisas sobre modelos mentais que buscam caracterizar através de um exame cuidadoso o modo de entendimento das pessoas em algum domínio de conhecimento [Gentner, 1983].

Em modalidades de ciências, como a Física, Química, Biologia, etc., tem-se diversos conceitos abstratos tais como equilíbrio, pressão, etc. Tais conceitos sempre são estudados via representações, geralmente na forma de dados tabulados obtidos experimentalmente, a partir dos quais se constrói gráficos ou se aplica fórmulas, tentando obter o entendimento do conceito. Estas representações gráficas podem ser vistas como modelos conceituais substitutos [Young, 1983], pois através delas podemos observar as características relevantes dos conceitos e de alguma forma fazer uma inferência sobre seu comportamento para dados não tabulados "a priori".

Pode-se utilizar o computador como auxiliar ao entendimento destes conceitos. A idéia é construir sistemas computacionais que forneçam representações alternativas dos conceitos e que possam funcionar em tempo real, ou seja, ao mesmo tempo em que um determinado experimento está sendo efetuado, tem-se um computador interligado aos instrumentos de medida de modo a poder fornecer representações alternativas, como descreve Tinker em [Tinker, 1985].

Dada a comprovada importância da representação, como um modelo conceitual, no entendimento e o fato de que não existe outra forma de entender conceitos e processos abstratos que não via representações, o que pretendo é verificar se o uso de sistemas computacionais de tempo real, que oferecem representações alternativas, auxiliam o entendimento de conceitos e processos abstratos. Para isso escolhi o contexto de programação de computadores, mais especificamente, programação utilizando a linguagem Logo.

6.4 A Escolha da Linguagem Logo

Como já mencionei em capítulos anteriores , simplicidade e visibilidade são duas características importantes de uma linguagem de programação para principiantes.

Visibilidade é relativa a métodos para visualizar partes selecionadas e processos da máquina notacional em ação. Uma das dificuldades de ensinar programação é descrever no necessário nível de detalhe a máquina que se está aprendendo a controlar.

Simplicidade implica em ter uma linguagem pequena, com poucas e bem definidas construções, restringindo as possíveis ações da máquina. Isto sem dúvida reduz a generalidade da linguagem, mas se o domínio de ações é bem escolhido obtem-se um ambiente amigável onde pode ficar mais fácil aprender programação.

O subconjunto gráfico da linguagem Logo que controla a Tartaruga é um exemplo desse tipo de ambiente, pois com uma pequena sequência de instruções obtem-se programas com efeitos gratificantes. Um conjunto de mais ou menos dez instruções e mais a estrutura de procedimentos e sub-procedimentos torna possível criar um rico e interessante, mas claramente limitado, conjunto de ações da máquina.

A natureza procedural do Logo e sua nomeação irrestrita permite que resultados interessantes sejam obtidos rapidamente por principiantes escrevendo programas simples. Esta simplicidade lógica é obtida através do casamento da linguagem com problemas familiares e de interesse do principiante.

Considerando-se estas características importantes ao principiante e mais o fato de que um dos objetivos da criação do

Logo foi o de ser uma linguagem facilitadora ao aprendizado de programação, a escolha de Logo justifica-se. Isto porque com todas estas características facilitadoras muitos problemas ainda são detetados, como foram analisados no Capítulo 4.

O avanço da parte gráfica para a parte de manipulação simbólica tem-se sérias barreiras, pois o nível de abstração cresce consideravelmente. Não se pode mais acompanhar a execução do programa. Esta perda da visibilidade tão marcante na parte gráfica, traz sérias dificuldades no avanço em programação, descaracterizando um pouco Logo como linguagem facilitadora.

Uma das razões porque pessoas acham programação simbólica tão dificil é que ela exige habilidade de visualizar procedimentos. A enorme quantidade de detalhe contida nos sucessivos estados pelos quais um programa passa, precisa ser mantida na cabeça do programador. Consequentemente, recursos utilizados no sentido de ajudar o usuário a visualizar a operação do programa são muito úteis no sentido de facilitar à atividade de programação.

Considerando-se estes aspectos, como ampliar o ambiente Logo,

de forma a facilitar esta passagem do ambiente procedural gráfico para o funcional, de processamento simbólico ?

A idéia é fortalecer o ambiente de modo a facilitar esta passagem. O que se observa é que enquanto se está no concreto, movimentando a Tartaruga, não existem problemas sérios. Mas quando são introduzidos conceitos abstratos como recursão, passagem de valor via operações, passagem de parâmetros, etc., o Logo apresenta fraquezas como ambiente facilitador ao aprendizado de programação simbólica.

E respeitando as características de Logo como ambiente de programação, não se pretendeu construir um sistema que ensinasse nos moldes dos tutoriais, ou então, que fizesse uma depuração inteligente procurando identificar fontes de falhas conceituais. Ao invés disso, foi criado um sub-ambiente onde através da observação de representações dinâmicas da execução de um programa se pudesse abstrair "idéias poderosas", ou seja, tornar mais rico um ambiente de aprendizagem.

Através de um pequeno conjunto de representações se pretende instrumentar o aluno de forma a que ele possa construir um modelo

mental adequado do funcionamento da linguagem, ao mesmo tempo em que lhe é fornecido meios de estruturar este novo domínio de conhecimento.

A forma escolhida de como propor uma ferramenta computacional para auxiliar o entendimento de conceitos computacionais, sem dúvida tem raízes na "cultura Logo", mas não está restrita à linguagem Logo, podendo ser utilizada, enquanto modelo, em qualquer outro contexto de programação.

Pode-se estender a idéia à outras áreas de conhecimento, onde se deseje dar visibilidade através de representações computacionais, a processos e conceitos abstratos.

6.5 Metodologia

A descrição da metodologia de trabalho será dividida em duas partes, a primeira abordando como foi definido o sistema computacional e a segunda, qual o método de observação de uso adotado.

6.5.1 Método de definição do sistema

Foi definida a construção de um ambiente que apresentasse múltiplas representações da execução de um programa escrito em Logo e que estas representações enfatizassem aspectos de programação tais como escopo de variáveis, controle de fluxo de execução, chamada e retorno de procedimentos, etc....

Foi feita a opção de representar a execução de um procedimento não a nível de comando ou expressões, e sim ressaltar estruturas globais e dar bastante ênfase a parte de controle de fluxo. Esta decisão foi tomada por estar interessada em auxiliar a aquisição de três conceitos computacionais - variáveis, fluxo de execução e recursão - que julgo fundamentais. Defini o que podemos denominar representações a nível macro, que enfatizavam aspectos da execução que julgo essenciais para o entendimento destes conceitos.

O problema que se seguiu foi definir quais seriam as representações a serem utilizadas e qual a formato geral de apresentação do sistema.

Tradicionalmente, ou seja, no modo "giz e quadro-negro" de ensino, são utilizadas representações da execução de um programa. Elas são feitas com o objetivo de ajudar a entender aspectos de programação que estão muito ligados ao funcionamento do programa quando em execução. Fiz então uma análise destas representações com o intuito de extrair características que seriam de interesse aos objetivos do sistema.

A opção essencial foi por simplicidade de apresentação. Utilizei a técnica de dividir a tela em múltiplas janelas, onde em cada uma está ressaltado um aspecto da execução. A simultaneidade da apresentação é que garante o entendimento de conceitos que dependem de aspectos complementares, mostrados em janelas diferentes.

Outro aspecto que foi considerado essencial, foi o de garantir o máximo de interatividade entre o usuário e o sistema. O usuário tem absoluto controle sobre o processamento. Foram adotadas técnicas de interação existentes na maioria dos sistemas computacionais modernos e ditos amigáveis.

Definido e implementado o sistema partiu-se para a fase de observação do uso.

6.5.2 Método de Observação

O método de observação escolhido foi o clínico estruturado descrito por Perkins em [Perkins, 1983].

Junto ao sistema, o experimentador sentava com o estudante e apresentava um problema de programação. O experimentador observava como o estudante tentava resolver o problema, solicitando ocasionalmente por explicações das intenções e idéias. Quando o estudante encontrava alguma dificuldade substancial que impedia seu avanço, o experimentador intervinha dando idéias, tanto para ajudar o estudante, como para descobrir a natureza da dificuldade.

Trabalhando com linhas gerais pré-definidas o experimentador tentava encontrar pistas responsáveis pela atitude do estudante em uma determinada situação. O experimentador tomava notas das coisas que o estudante dizia e recolhia todo e qualquer material escrito que o estudante produzia, além de gravar os passos

intermediários da solução do problema, na forma de códigos de procedimentos intermediários.

Depois disso o conjunto de dados era analisado de modo a poder delinear o pensamento do estudante e interpretar a natureza das dificuldades que ele havia encontrado.

Todo esforço foi concentrado em observar como o aprendiz desenvolvia um modelo de funcionamento de um programa e mais geral, o modelo de funcionamento do computador a nível de programa. Tendo isto em vista, na maioria das vezes era solicitada explanação do estudante sobre o que um programa fazia e dali eram encaminhadas alterações no programa a serem efetuadas pelo estudante de modo a verificar o nível de entendimento do modelo que o estudante havia construído.

Eram feitas muitas atividades fora do computador solicitando ao estudante que descrevesse o funcionamento de um código, e depois ir observar o funcionamento junto ao sistema.

Os problemas eram bastante simples, para que dificuldades com conceitos computacionais não fossem ofuscadas com dificuldades de

solução de problemas, ou seja, com dificuldades em explicitar o método de solução de um problema.

Como estava interessada em verificar a evolução dos modelos mentais que estudantes formam sobre o processamento de programas, ou seja, sobre o funcionamento do computador, a escolha deste método de observação foi decorrência natural.

Através deste ambiente clínico de observação pretendia ressaltar a característica básica do uso de modelos mentais, que é a de fornecer justificativas e explanações, além do aspecto de predição de resultados que implica na "execução" do modelo mental. Levantando estes aspectos posso verificar as alterações que vão ocorrendo nos modelos criados e a influência do uso do sistema nestas transformações. Além disso, o ambiente cliníco é demonstradamente um bom ambiente de aprendizagem [Ackermann, 1991].

Esta metodologia foi utilizada durante toda a fase experimental de meu trabalho. Os resultados estão relatados em três próximos capítulos. Vinte e três pessoas foram observadas,

divididas em três grupos. Para cada grupo foi definido um objetivo dentre os a seguir:

. diagnosticar e remediar o conceito de recursão (Capítulo 8)

. aprendizado de recursão no processamento simbólico (Capítulo 9)

. verificar o quanto especialistas em computação dominam o conceito de recursão (Capítulo 10)

Capítulo 7

Descrição do Sistema Implementado

7.1 Introdução

O objetivo ao se implementar o sistema computacional que descrevo neste capítulo não foi o de obter um interpretador para a linguagem Logo e sim, um ambiente de representações de um programa em execução.

Para que isso fosse possível foi necessário escrever um protótipo de interpretador para um restrito subconjunto de comandos da linguagem Logo, visto não existir nenhuma implementação disponível no mercado nacional para micros da linha PC. Dado este fato tentou-se escrever um núcleo de interpretador que pode ser facilmente expandido de forma à tratar todos os comandos da linguagem Logo.

A idéia que norteou a implementação foi a de utilizar o sistema como um apêndice ao sistema Logo do MSX, no sentido de se poder visualizar a execução de procedimentos editados no MSX.

A interação do usuário com o interpretador é via uma tela básica onde são apresentadas representações da execução de um programa.

A seguir é feita, em duas partes, uma descrição detalhada do sistema implementado: a primeira trata especificamente dos aspectos relativos as representações do programa em execução com as quais o usuário interage e mais um exemplo completo de uso do sistema; a segunda trata de aspectos relativos à forma como foi implementado o sistema, principalmente quanto à estrutura de dados utilizada. Acompanha a descrição um exemplo completo de uso do sistema.

7.2 Apresentação do sistema

A forma de interação do usuário com o sistema ou então, sua apresentação pode ser vista como possuindo duas formas básicas de representações que denomino de locais e globais.

O subconjunto de representações locais tem como objetivo mostrar num certo ponto de execução o que está acontecendo e o que o computador "conhece" a respeito do programa. Perde-se nestas representações a história passada da execução, ou seja, não é visível o processo que conduziu aquele momento da execução. Por acreditar que visualizar de alguma forma o processo como um todo é importante para o entendimento dos conceitos abordados,

principalmente com relação aos aspectos de chamada e retorno de procedimentos e programas com muitos níveis de execução, definiu-se outra representação que tenta mostrar estes aspectos da história de execução do programa. Esta outra representação denomino de global.

Quanto à representação local, o sistema proporciona uma tela subdividida em três janelas, como pode ser visto na **figura 1.**

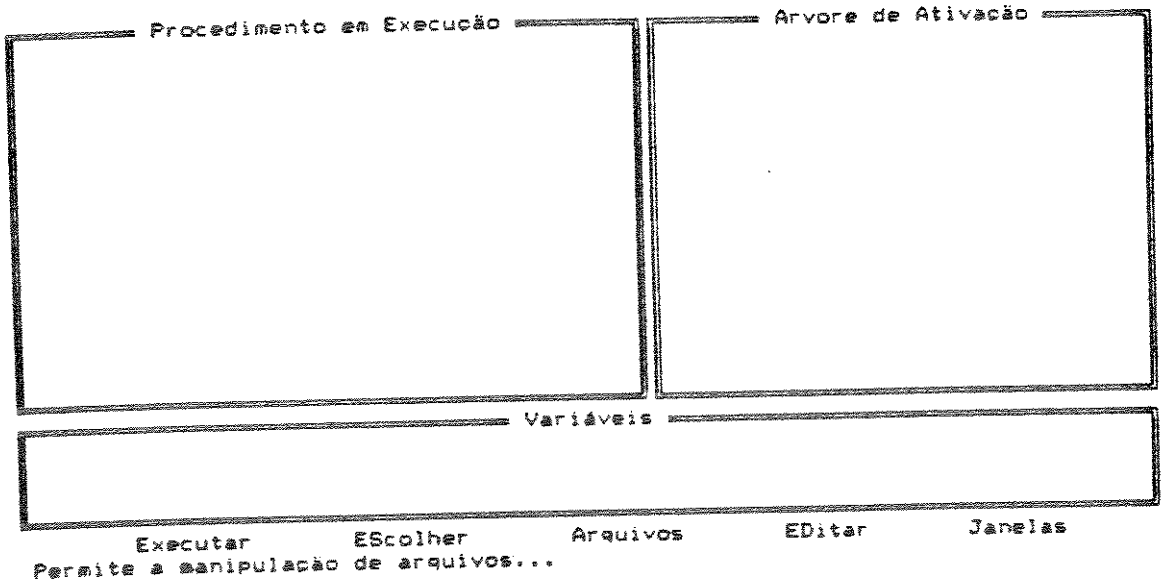


figura 1

A primeira janela, nomeada "Procedimento em Execução", contém o texto do procedimento em execução e o cursor em vídeo reverso, ou em outra cor se o vídeo for colorido, mostra a instrução que está sendo executada. Através desta representação pretende-se deixar claro, dentre outros aspectos, a sequencialidade de execução das instruções. Este é um aspecto que pode parecer irrelevante mas que ocasiona muitos problemas quando da construção de procedimentos com muitas opções condicionais, com estruturas repetitivas e fundamentalmente, com muitos sub-procedimentos. Observa-se que para muitos programadores, não é clara a sequência em que são executadas as instruções no retorno de sub-procedimentos.

No sistema quando ocorre a chamada de um sub-procedimento, o texto do procedimento chamante é substituído pelo do procedimento chamado, e uma mensagem informa que está ocorrendo uma chamada. O mesmo ocorre quando do retorno do sub-procedimento, onde uma mensagem informa que está ocorrendo um retorno e volta à tela o texto do procedimento chamante e a execução prossegue. O mesmo esquema é utilizado no caso de procedimentos recursivos. Obtem-se então uma representação de como é efetuada a chamada e retorno de procedimentos.

Na segunda janela, nomeada "Arvore de Ativação", é representada a estrutura dinâmica de chamadas de procedimentos. Nesta janela para cada chamada de procedimento é criado um retângulo com nome do procedimento chamado e que contém o nome e valor dos parâmetros no momento da chamada. Estes retângulos vão sendo construídos e eliminados de acordo com a ocorrência de chamada ou retorno da chamada de procedimentos.

A terceira janela, nomeada "Variáveis", contém o nome das variáveis ativas e de seus valores correntes. Cada alteração do valor de uma variável dentro do programa é refletida nesta janela com realce. O objetivo desta terceira janela é ressaltar os aspectos relativos à conceituação de variáveis e seu escopo de validade, mostrando variáveis que "passam a existir" no momento da chamada de um procedimento e que "deixam de existir" no retorno.

Pode ser observado que além das janelas, existe um menu de opções que irá coordenar a execução do sistema. A seleção é feita via a letra que está em realce ou selecionando através das teclas de posicionamento do cursor.

Na **figura 1** o cursor está posicionado na opção Arquivos. Caso haja a seleção desta opção aparece a tela mostrada na **figura 2.**

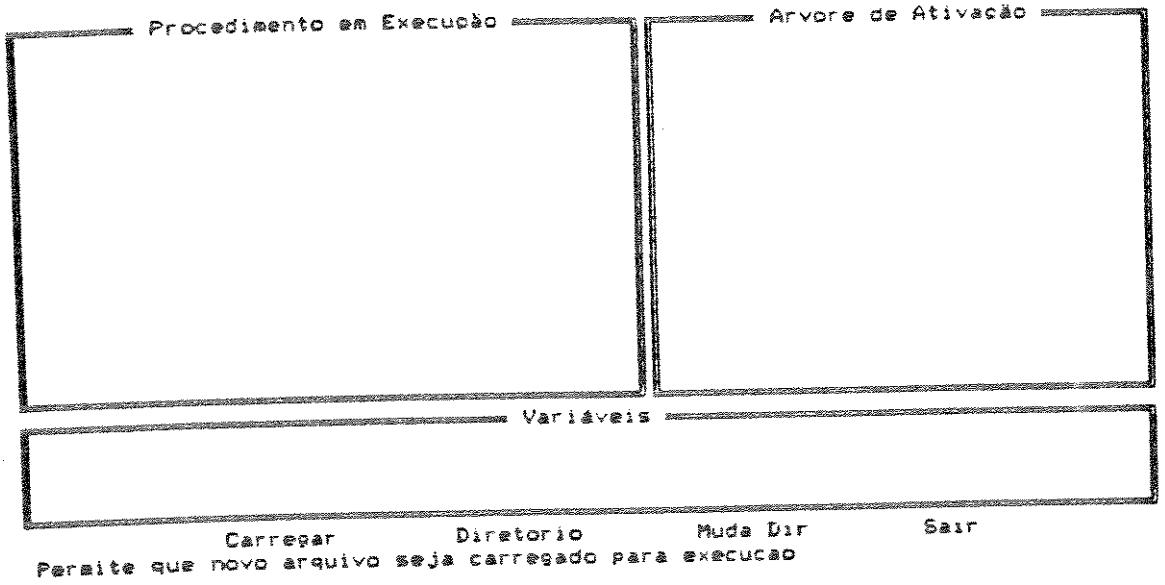


figura 2

A tela é a mesma somente as opções de operações possíveis são diferentes. Aparece a opção Carregar que irá permitir que um novo arquivo seja carregado para execução. Esta é a opção na qual o cursor está posicionado na **figura 2.**

A opção **Diretório** irá permitir a visualização do diretório corrente, como é mostrado na **figura 3.** A opção Muda Dir permite

que se altere o diretório corrente e a opção Sair permite a saída do sistema, retornando ao sistema operacional.

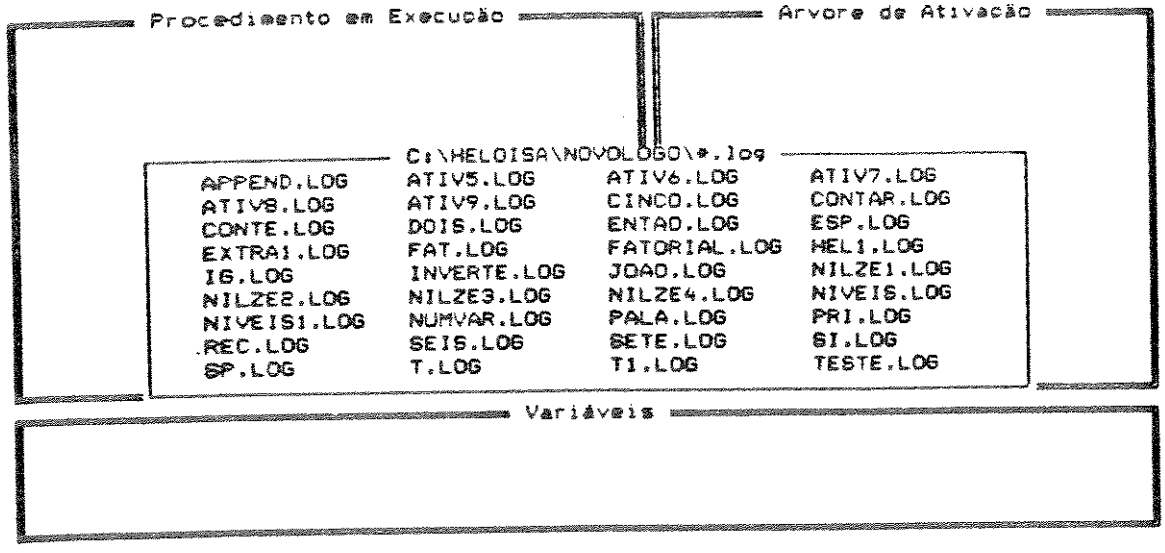


figura 3

Na tela inicial (figura 1) existem ainda as opções **EScolher**, **EDitar** e **Janelas**.

A opção **EDitar** dá entrada ao editor do sistema, que ainda é uma implementação provisória, pois não fazia parte da definição original do sistema. Pode ser acionado a qualquer momento da execução.

O editor foi implementado para agilizar o processo do usuário desejar fazer pequenas alterações ou então escrever pequenos programas para testar rapidamente alguma hipótese sobre o funcionamento da linguagem. As telas que dão entrada no editor estão mostradas nas figuras 4 e 5.

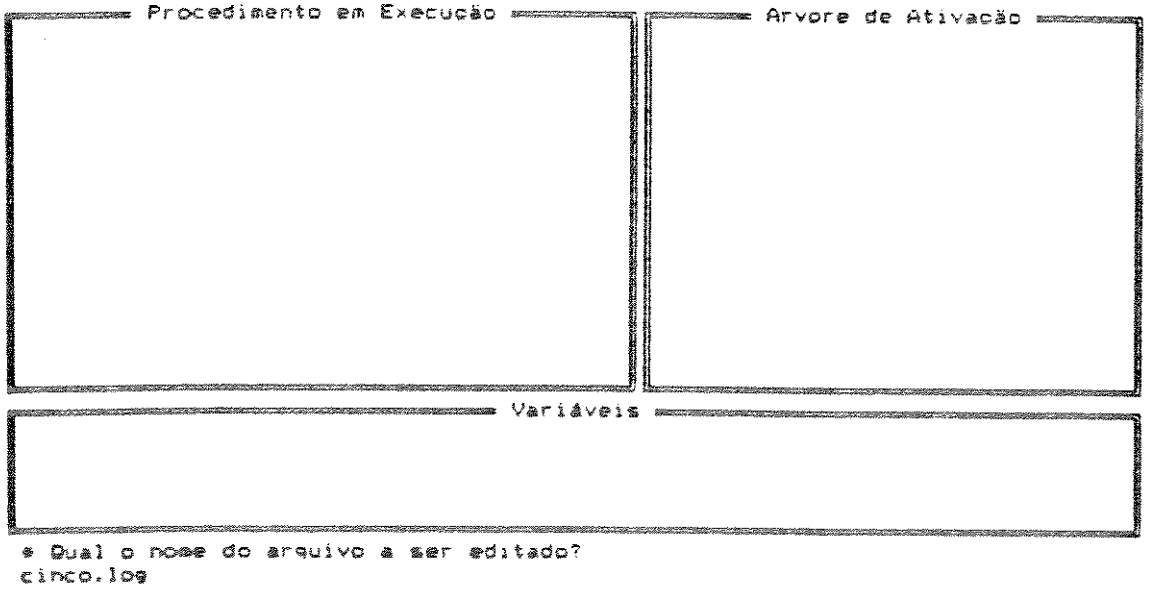


figura 4

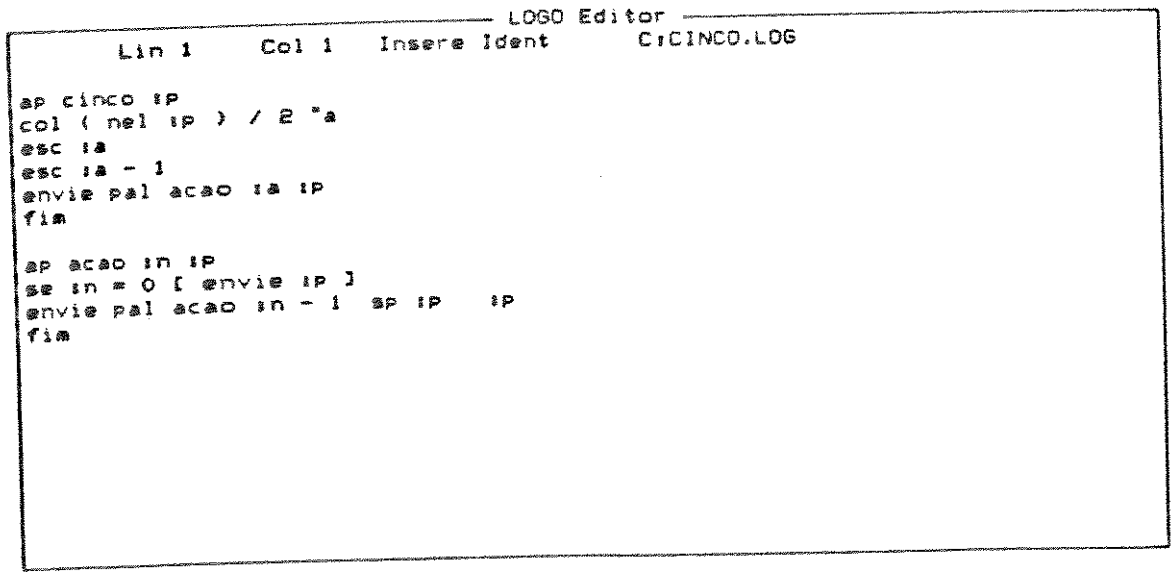


figura 5

A opção **Janelas** do menu inicial (**figura 1**)permite alterar o tamanho das janelas, caso haja interesse em focalizar a atenção em uma das representações. O uso desta opção está exemplificado nas figuras 6 e 7, onde está sendo aumentado o tamanho da janela de variáveis. O tamanho é alterado utilizando-se as teclas de movimento de cursor.

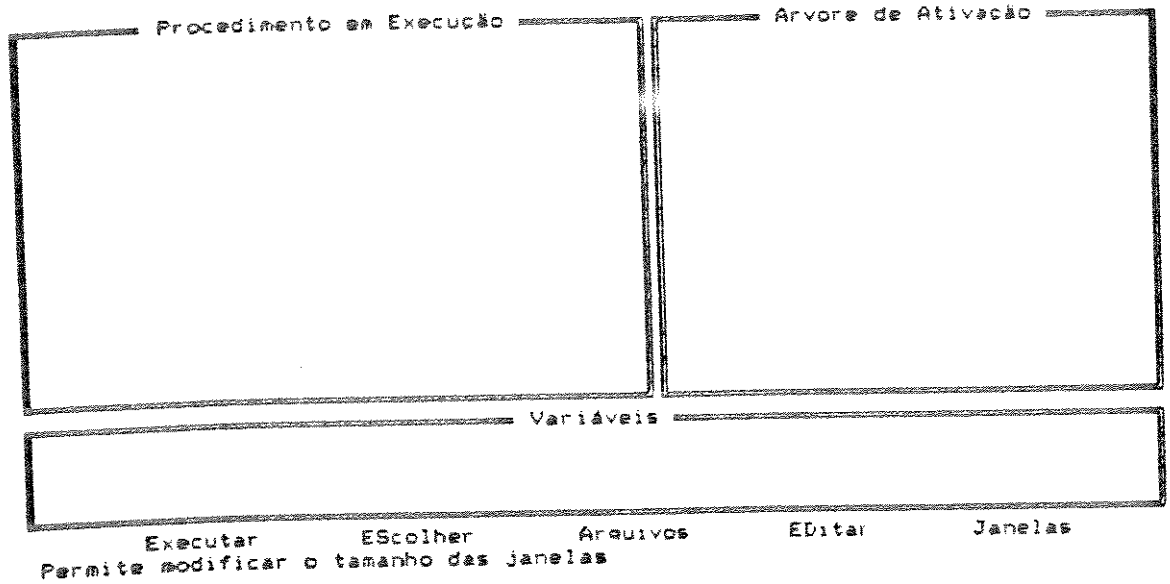


figura 6

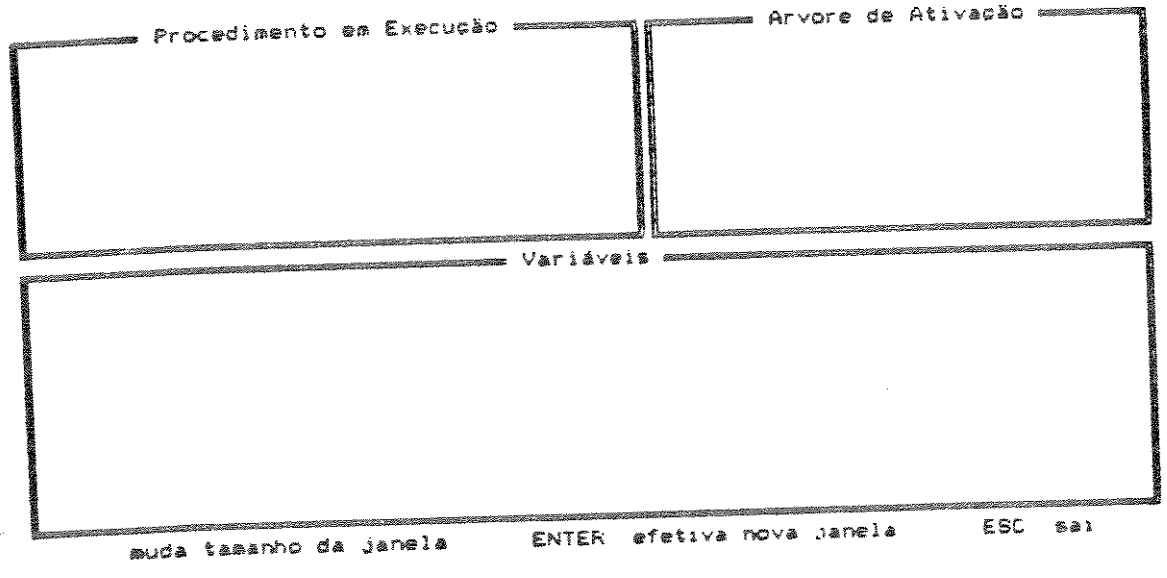


figura 7

A opção **EScolher** (figura 1) conduz ao processo de seleção do modo de execução desejado.

O arquivo pode ser executado de dois modos diferentes: **Automático** e **Passo-a-Passo**, como é mostrado na figura 8 que é a tela seguinte à seleção da opção **EScolher**.

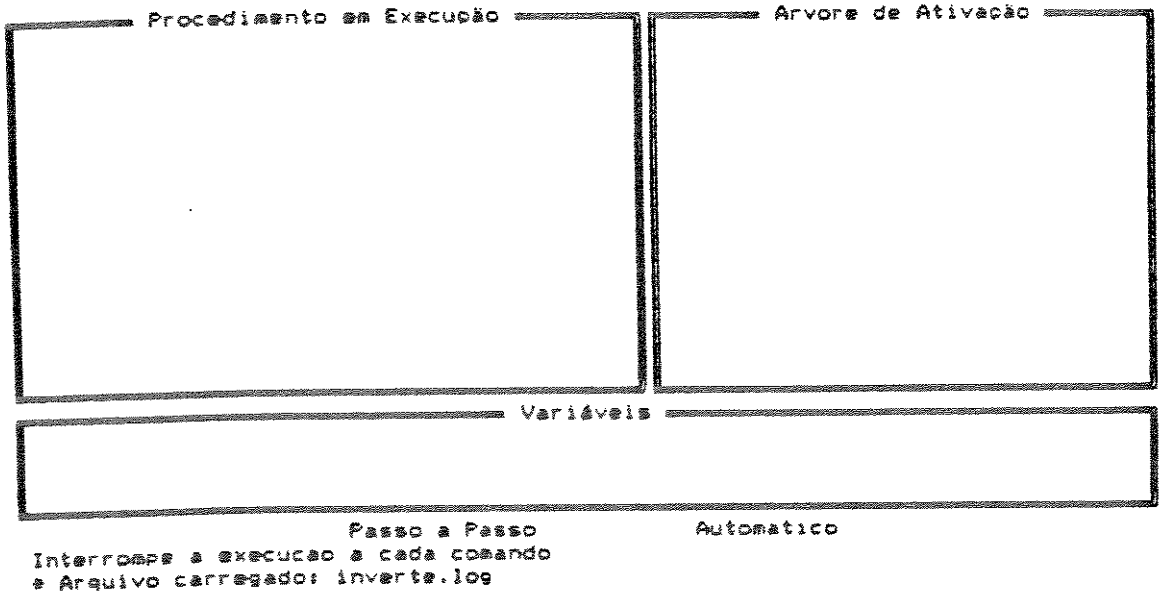


figura 8

No modo automático não é permitido qualquer tipo de interferência do usuário durante a execução e existem duas opções: **Mostrando** e **Direto** (figura 9). Dentro da opção **Mostrando**, são mostrados os passos da execução e na opção **Direto** a execução é feita como em um ambiente de programação qualquer, somente é apresentado o resultado final. São efetuadas paradas somente quando da entrada e saída de dados.

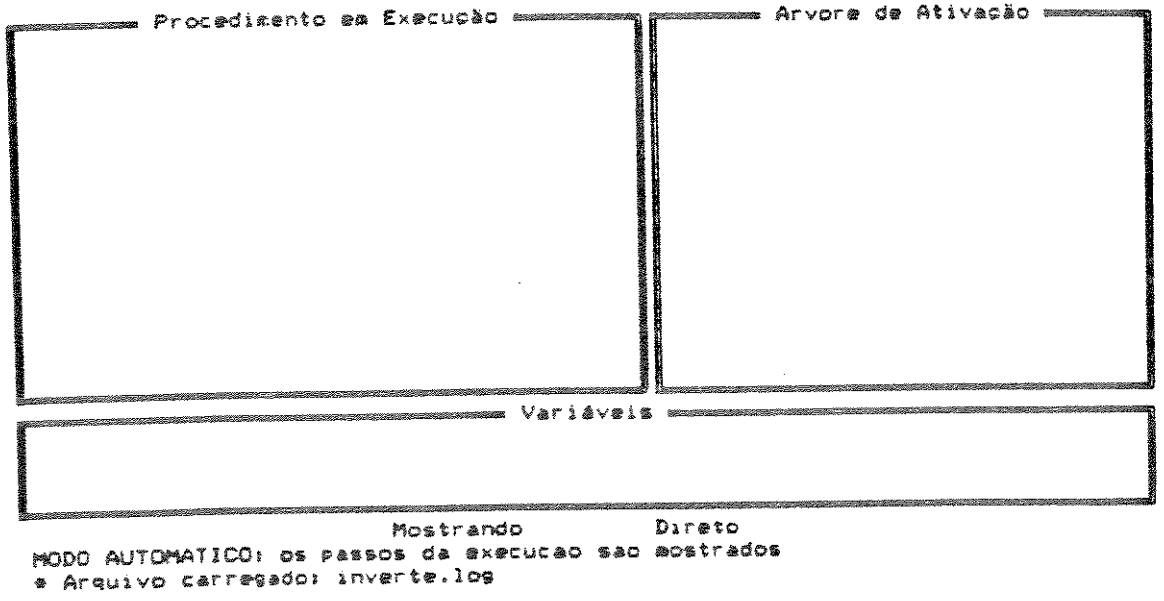


figura 9

O modo **Passo-a-Passo** também oferece duas outras opções: **Parando** e **Direto** (figura 10). A opção **Parando** deixa toda a velocidade de execução a cargo do usuário. Depois da execução de cada comando há uma parada e o reínicio da execução é controlado pelo usuário. A opção **Direto** a execução de cada comando provoca uma pausa fixa definida pelo sistema e não exige interferência do usuário a cada comando para prosseguir a execução como na opção **Parando**.

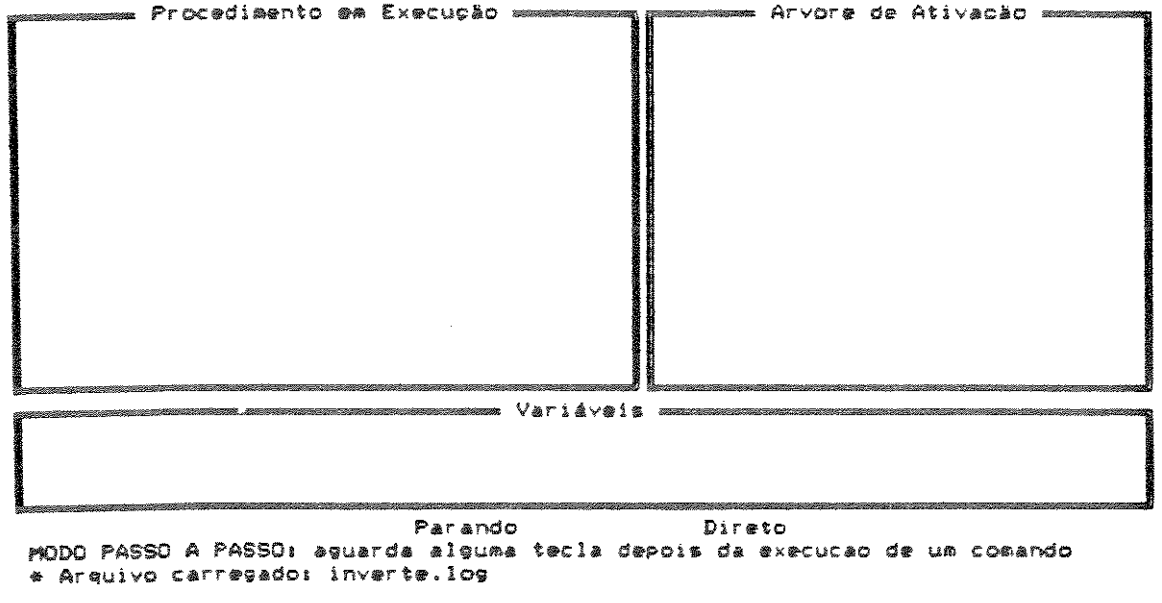


figura 10

Sempre ao posicionar o cursor em uma opção aparece uma mensagem que dá uma explicação breve da função da opção, de modo a faciltar a interação com o sistema. Isto pode ser observado em todas as figuras apresentadas.

Finalmente a opção **Executar** do menu inicial **(figura 1)** permite a execução do arquivo carregado de acordo com as opções de execução escolhidas. Ao ser selecionada esta opção aparece uma tela em que pode ser selecionada a opção **Piscando** e **Sem Piscar** **(figura 11)**. Isto possibilita que seja dado um realce maior a cada aletração que ocorre nas representações.

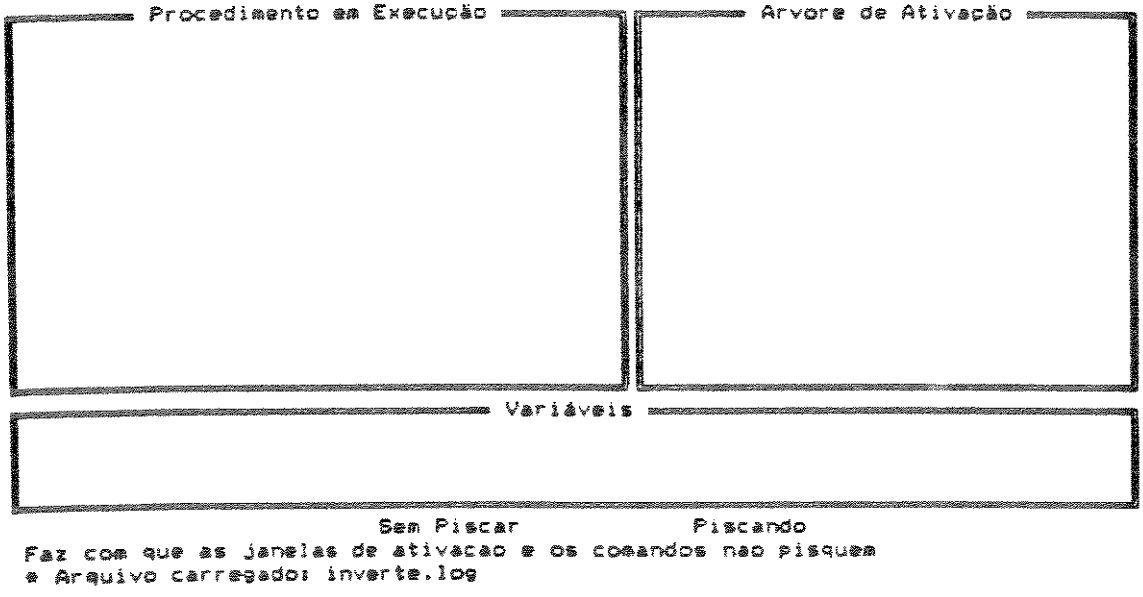


figura 11

Durante a execução de um procedimento, em todos os modos de execução, a menos do Automático e Direto, toda execução de comando é acompanhada de mensagens informando o que está sendo feito, como "retorno de chamada", "chamada de procedimento", "comando de saída", etc.

Existe uma quarta janela, nem sempre presente na tela, onde são pedidos valores de entrada e onde são mostrados valores de saída (figura 12). Esta janela somente é aberta quando da execução de comandos de entrada e saída e desaparece ao término da execução dos comandos respectivos ou quando o usuário desejar.

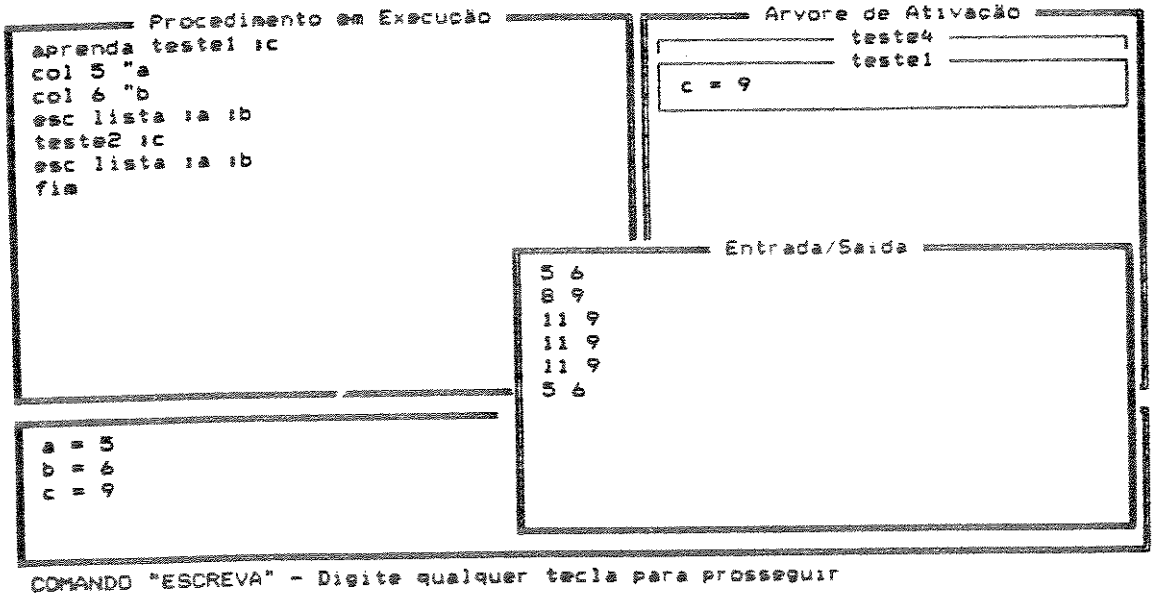


figura 12

Outra janela, que também não está sempre presente na tela, indica saídas relativas ao comando ENVIE (retorno de função), informando qual valor foi enviado, por qual procedimento e para qual procedimento (**figura 13**). Também esta janela somente aparece quando do retorno de uma operação e desaparece no momento em que é concluído o retorno ou sob comando do usuário.

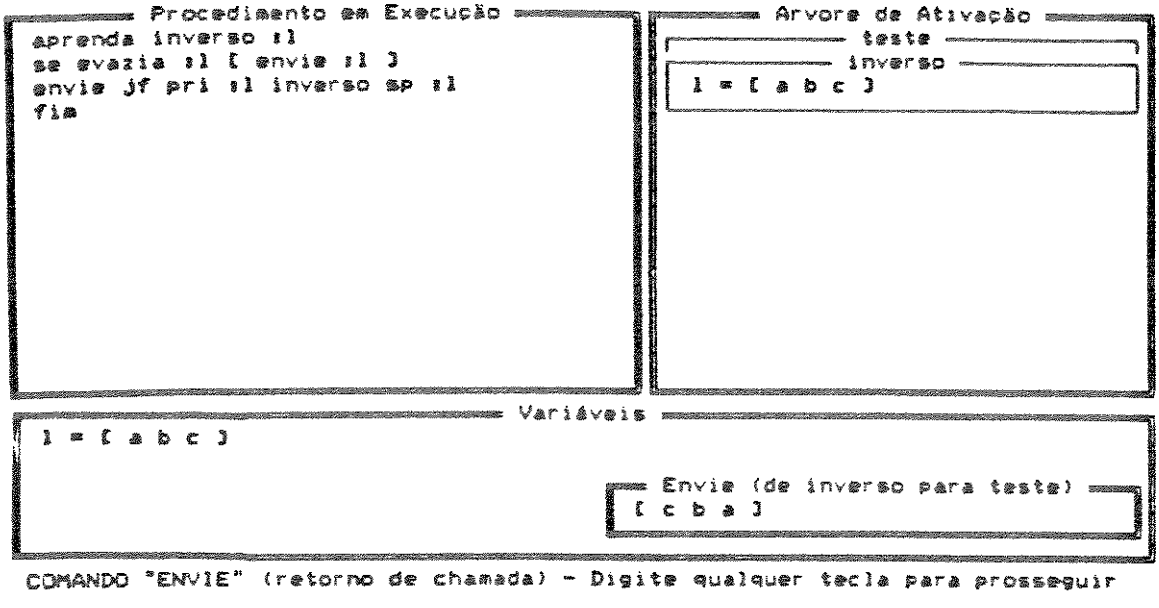


figura 13

Através de uma tecla de função pode-se ter acesso à representação global da execução. Esta representação é feita em tela única e mostra um esquema da execução do procedimento. Nesta tela tem-se o texto dos procedimentos que compõe a área de trabalho, ou seja, o arquivo utilizado na execução. Através de um esquema de cores são utilizados cursores que indicam qual instrução está sendo executada e quais as execuções em suspenso, ou seja, pontos de chamada e retorno nos outros procedimentos ativos da área de trabalho. Esta tela é mostrada na **figura 14**, que corresponde à representação global do arquivo que aparece na **figura 12.**

Arquivo niveis1.log

```
aprenda teste
col 1 "a
col 2 "b
teste1 5
teste1 :b
esc lista :a :b
fim

aprenda teste1 :c
col 5 "a
col 6 "b
esc lista :a :b
teste2 :c
esc lista :a :b
fim

aprenda teste2 :d
col 8 "a
col 9 "b
esc lista :a :b
teste3 :a
esc lista :a :b
fim
    PgUp/PgDn//  - rolam o texto          ESC  - volta ao contexto anterior
```

figura 14

Toda interação com o sistema é feita através da seleção de opções em menu. A qualquer momento da execução de um programa pode ser interrompida e se ter acesso a outro tipo de representação ou outro modo de execução.

Como não possui, no momento, características de um ambiente de depuração, o sistema somente trata programas sem erro de sintaxe não possuindo nenhum mecanismo implementado de detecção e recuperação de erros. Somente está prevista a deteção de alguns erros semânticos frequentes como, por exemplo, tentar extrair um elemento de uma lista vazia.

7.3 Exemplo de Uso

Nesta parte será apresentada uma sequência de telas que exemplifique a forma de uso do sistema.

Será mostrada a execução de um procedimento INVERSO, que inverte os valores de uma lista dada de entrada. A entrada e saída de valores é feita através de um procedimento chamado TESTE. Os procedimentos estão armazenados em um arquivo nomeado INVERTE.LOG listado a seguir.

```
aprenda teste
col line "1
esc inverso :1
fim

aprenda inverso :l
se évazia :l [ envie :l ]
envie jf pri :l inverso sp :l
fim
```

A **figura 15** apresenta a tela inicial do sistema e a seleção da opção **Arquivo** que dará entrada a tela apresentada na **figura 16.**

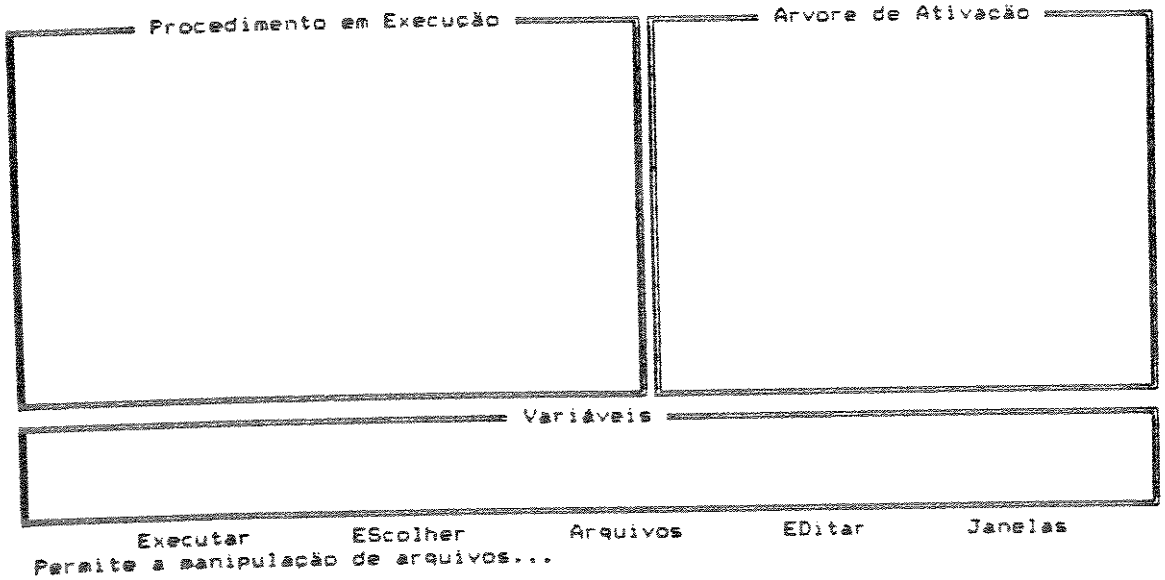


figura 15

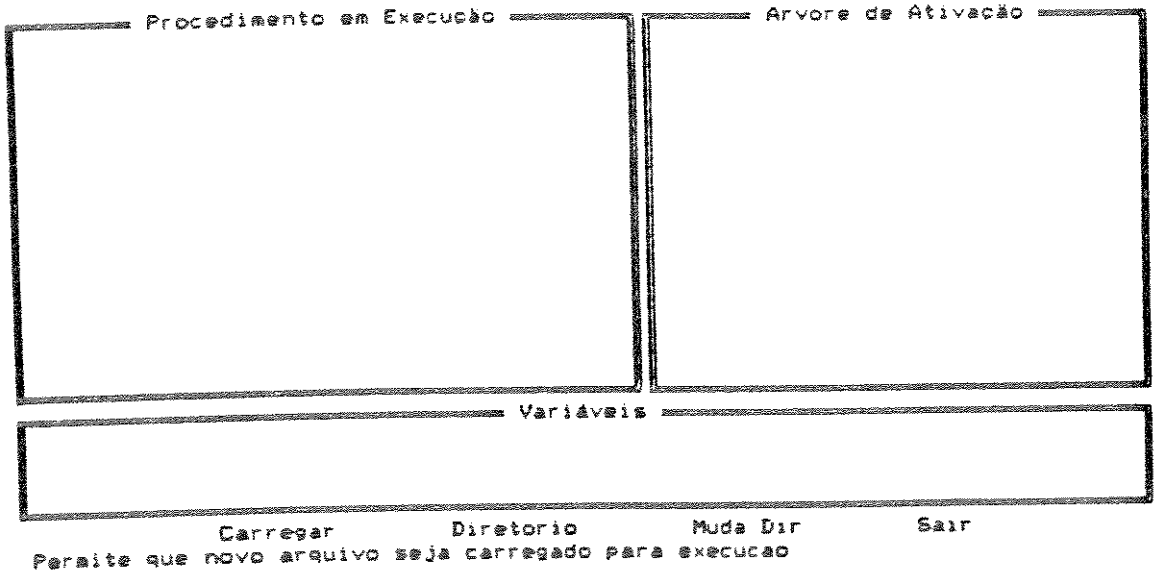


figura 16

Em seguida é selecionada a opção Carregar. O nome do arquivo desejado é informado e em seguida é solicitado o nome do procedimento inicial a ser executado. Estas telas são mostradas nas figuras 17 e 18.

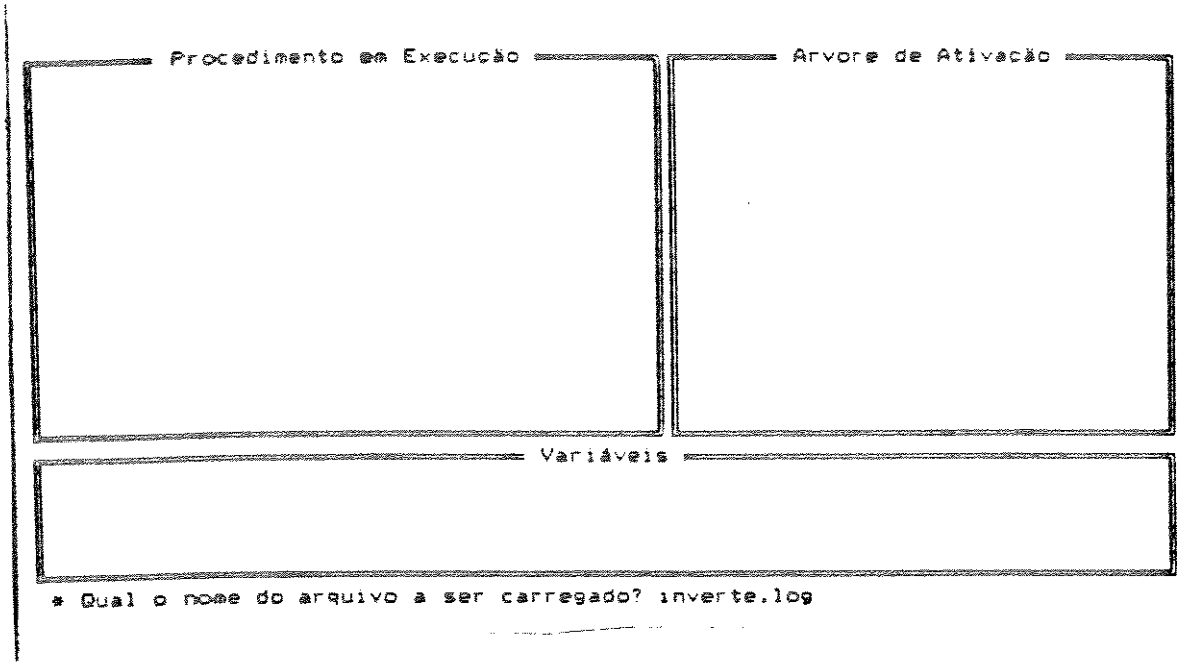


figura 17

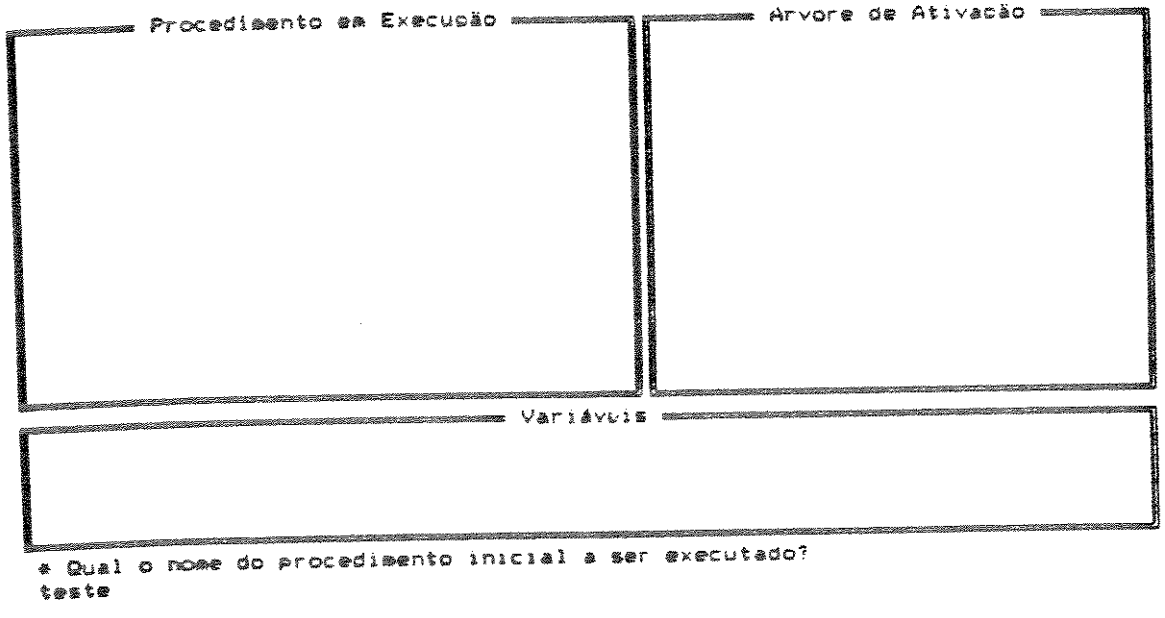


figura 18

Depois disso aparece novamente a tela inicial, sendo então selecionada a opção EScolher para fazer a escolha dos modos de execução. Neste exemplo irão ser selecionadas as opções: Passo-a-Passo, Parando e Sem Piscar. A sequência de telas é mostrada nas figuras 19, 20, 21 e 22.

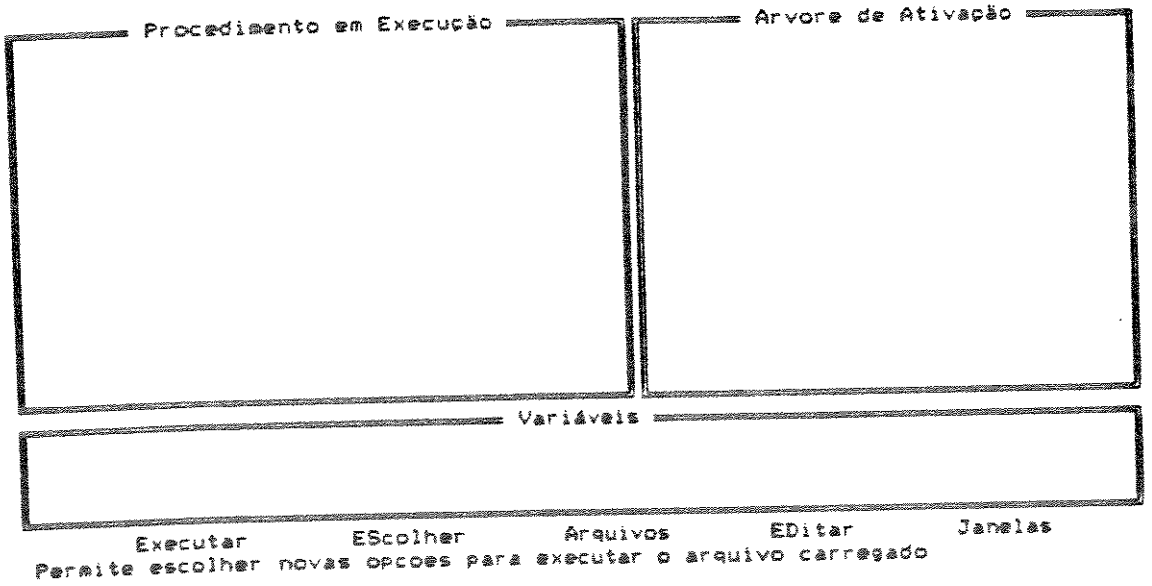


figura 19

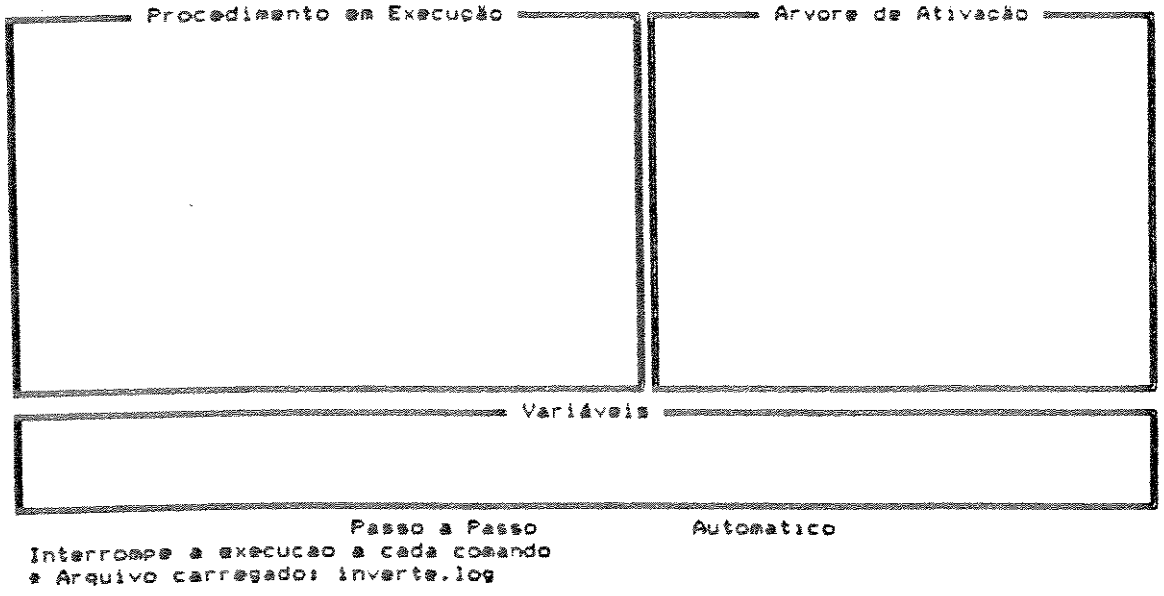


figura 20

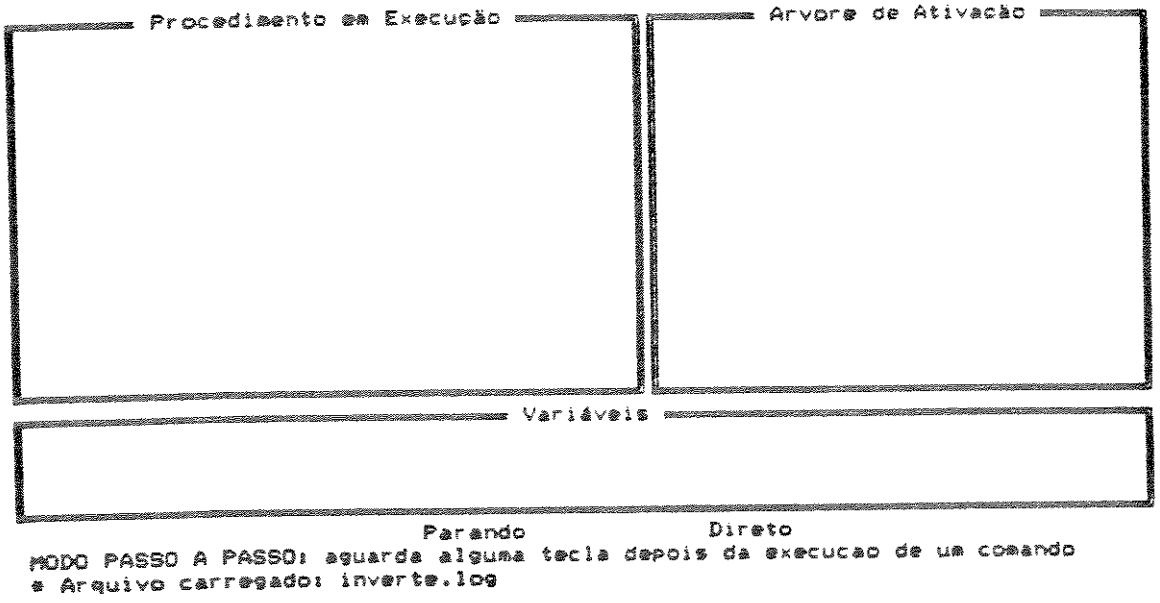


figura 21

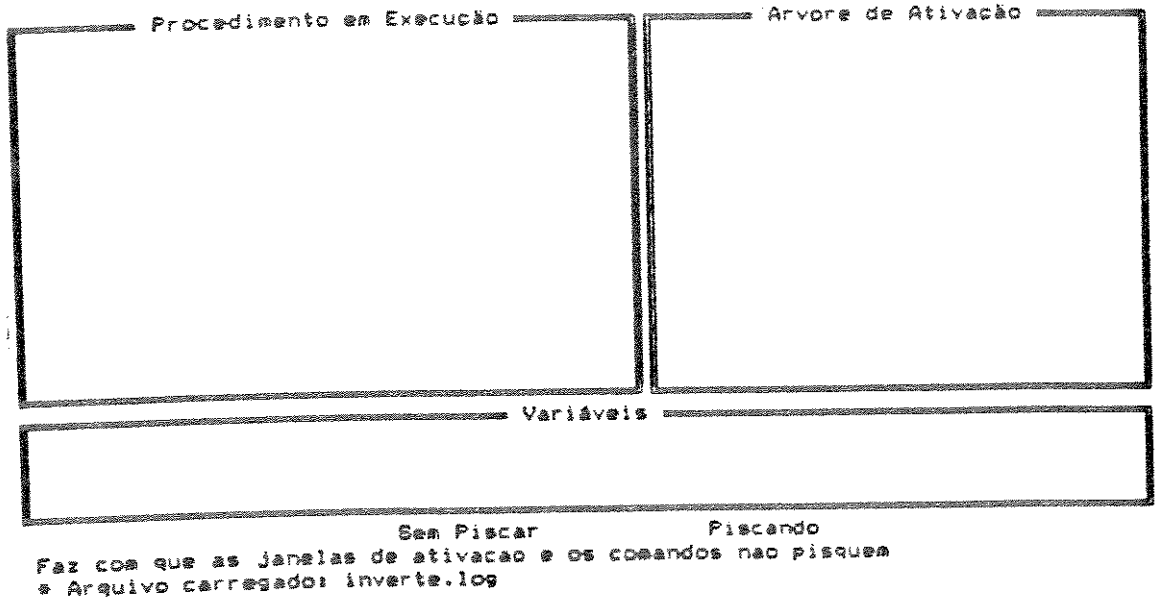


figura 22

Após a seleção do modo de execução, retorna a tela inicial onde é selecionada a opção **EXecutar** e entra-se em modo de execução. Irá aparecer o procedimento que está sendo executado inicialmente, no caso o procedimento teste e a árvore de ativação correspondente a esta primeira chamada. A tela desta fase está na **figura 23.**

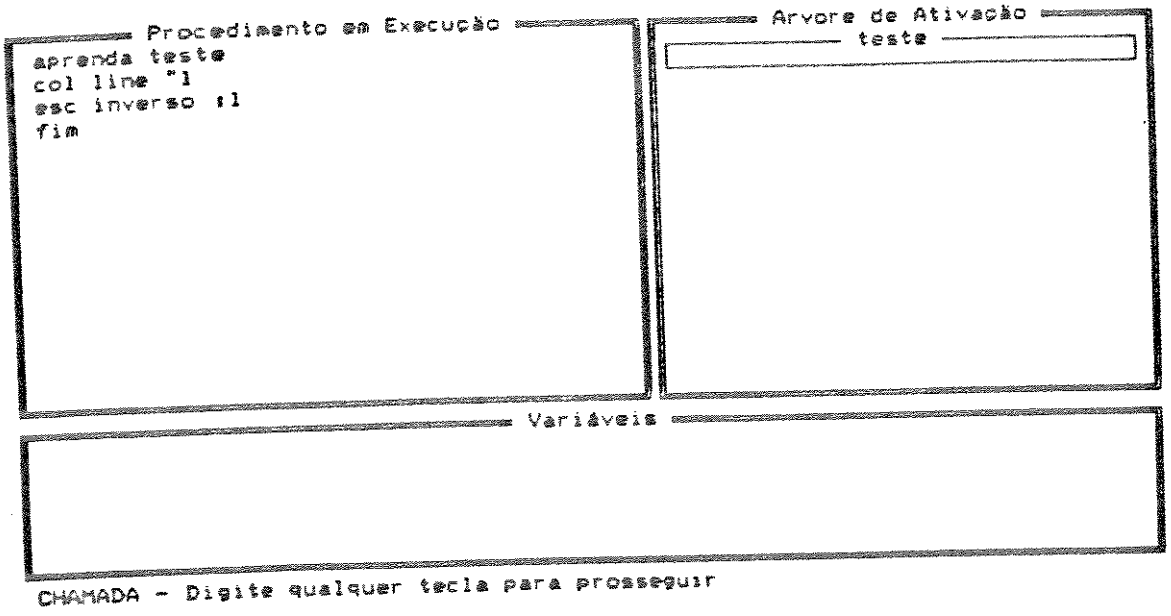


figura 23

Inicia a execução do procedimento teste, e é executado o comando **col line "l**. Este comando é de entrada, e então é aberta a janela de Entrada e Saída que irá permitir que se faça a entrada dos valores da lista, que será armazenada na variável **l**. As **figuras 24 e 25** mostram a execução deste comando.

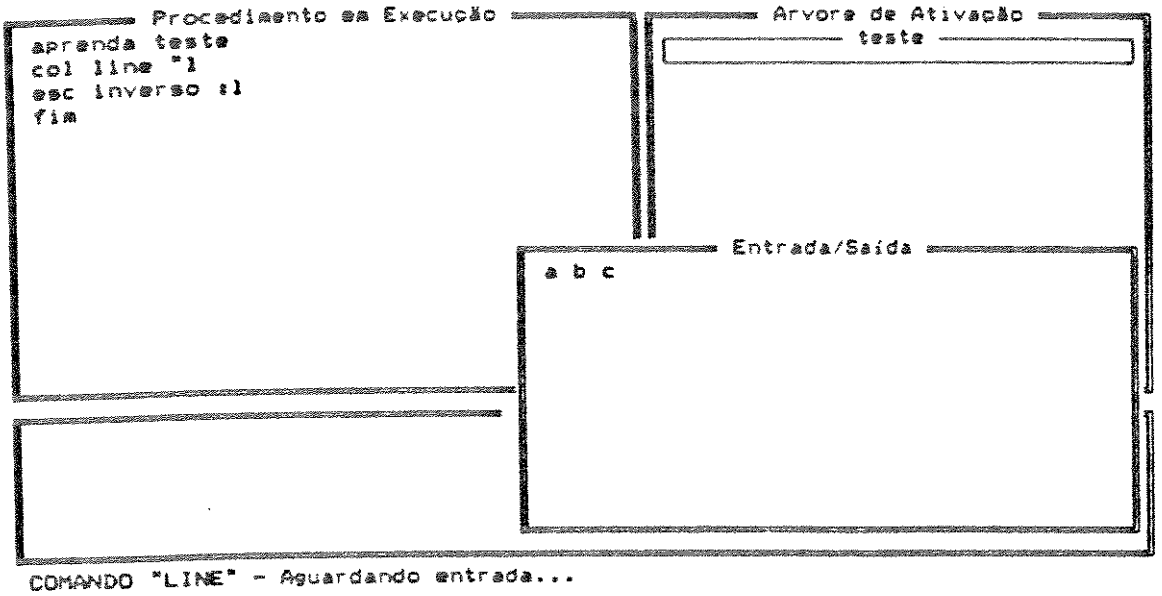


figura 24

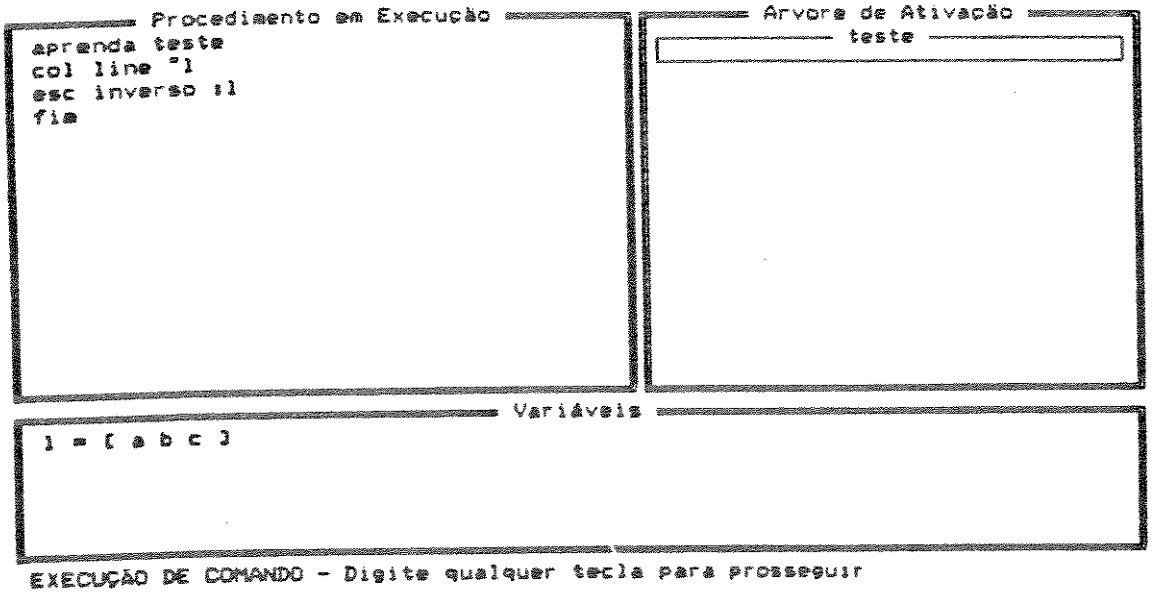


figura 25

A sequência de figuras de 26 a 30, mostra a execução do procedimento inverso em todos os passos da recursão, antes de iniciar os retornos.

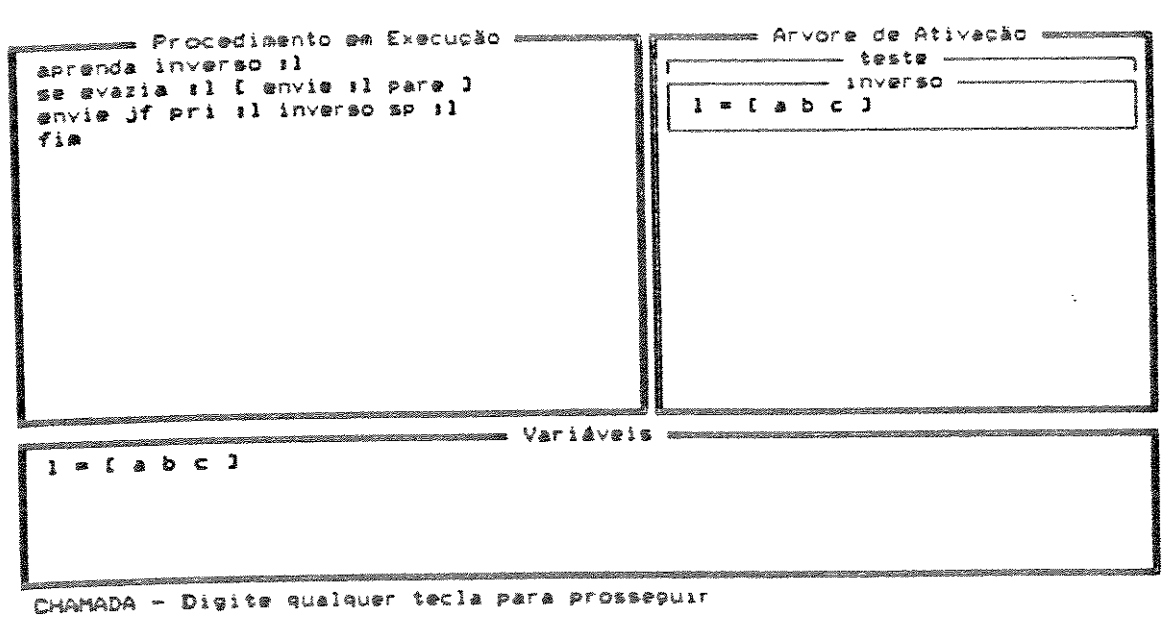


figura 26

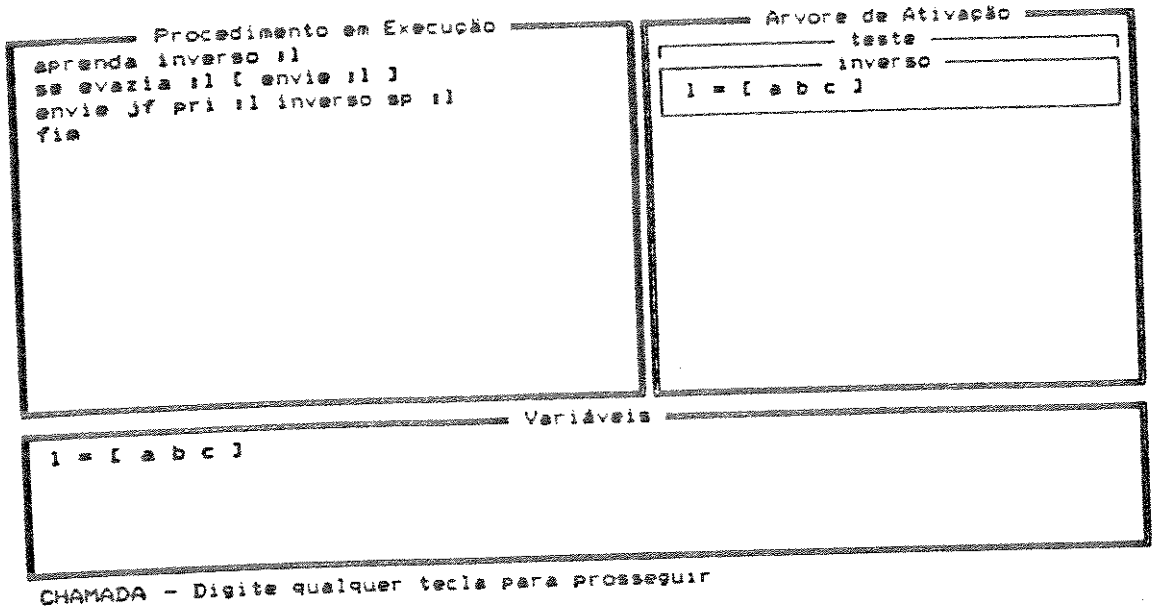


figura 27

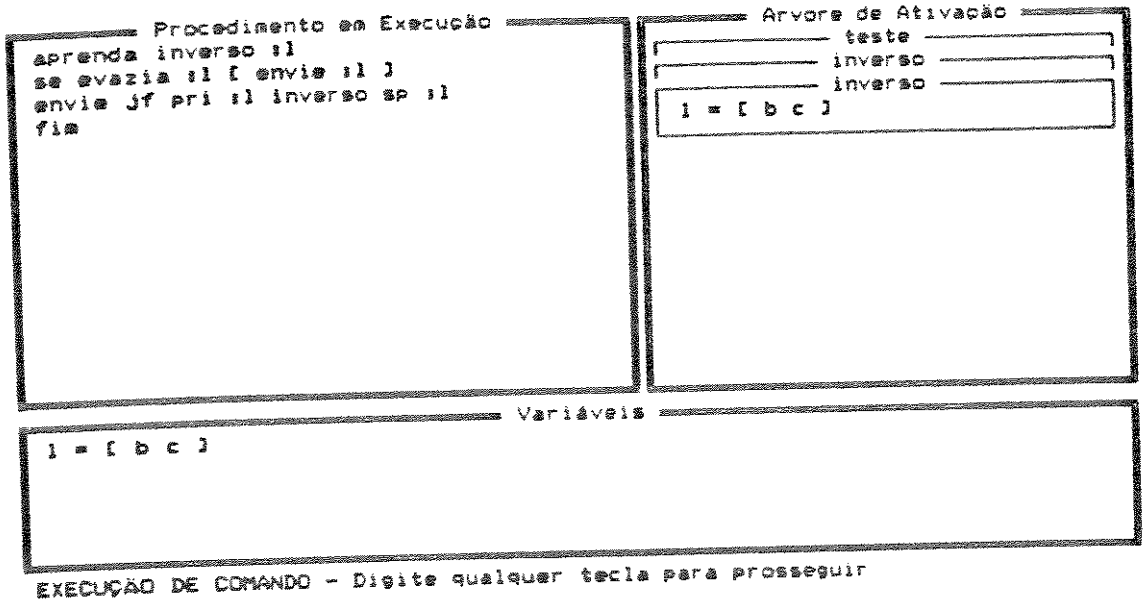


figura 28

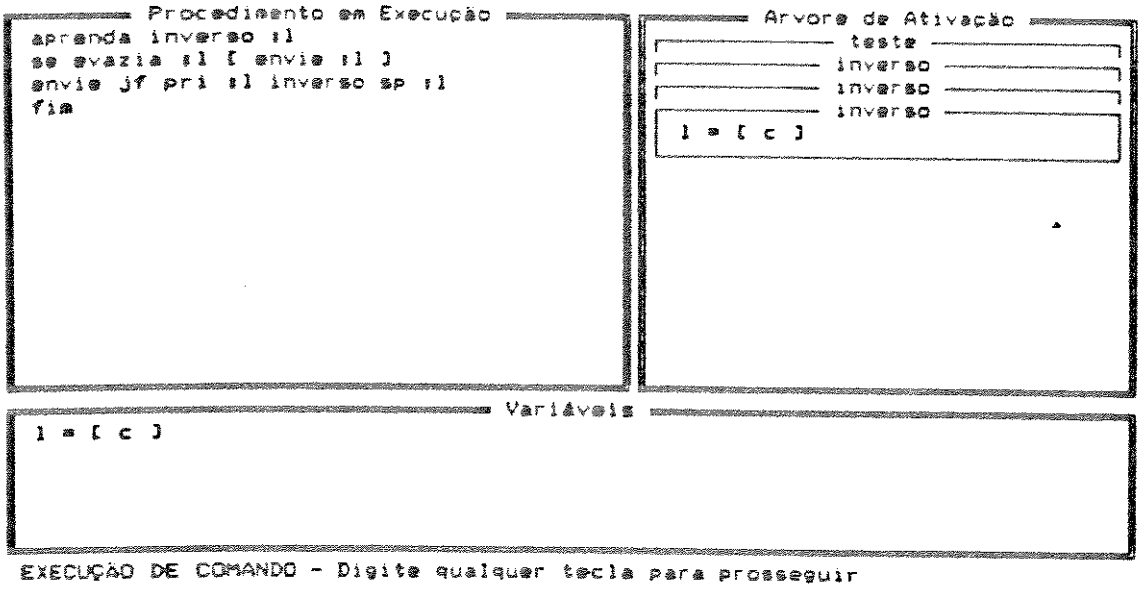


figura 29

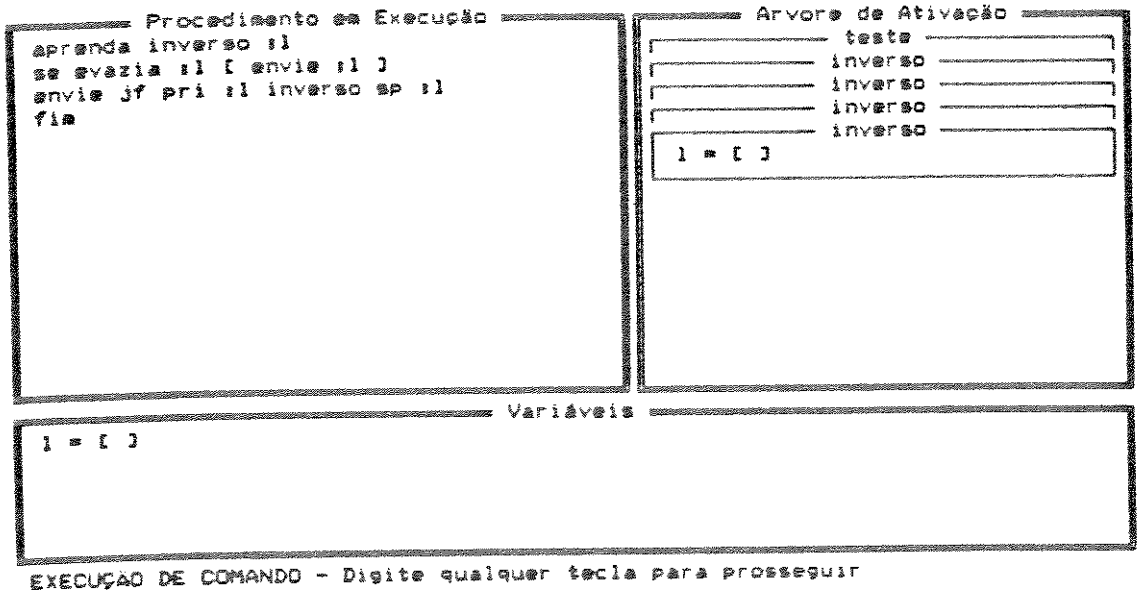


figura 30

E a sequência de figuras de 31 até 34 mostram as fases de retorno, onde são enviados os valores de retorno, que no caso são as fases de construção da lista invertida. Aparece a janela ENVIE, onde são especificados o valor a ser enviado, de qual procedimento e para qual procedimento. Observe que no último passo, o valor é enviado para o procedimento teste que inicialmente chamou o procedimento inverso.

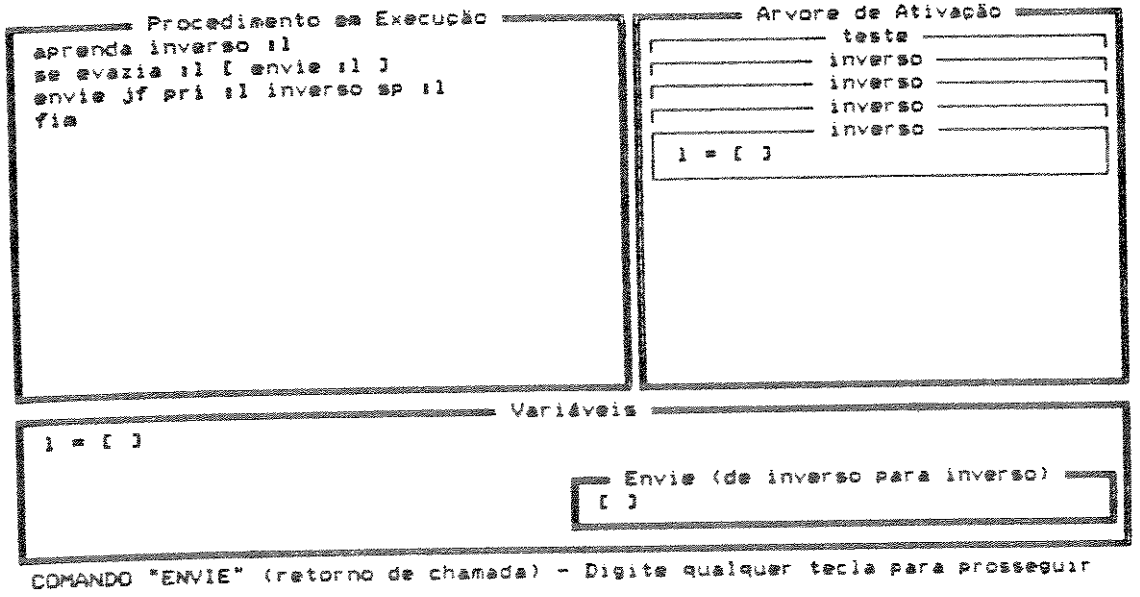


figura 31

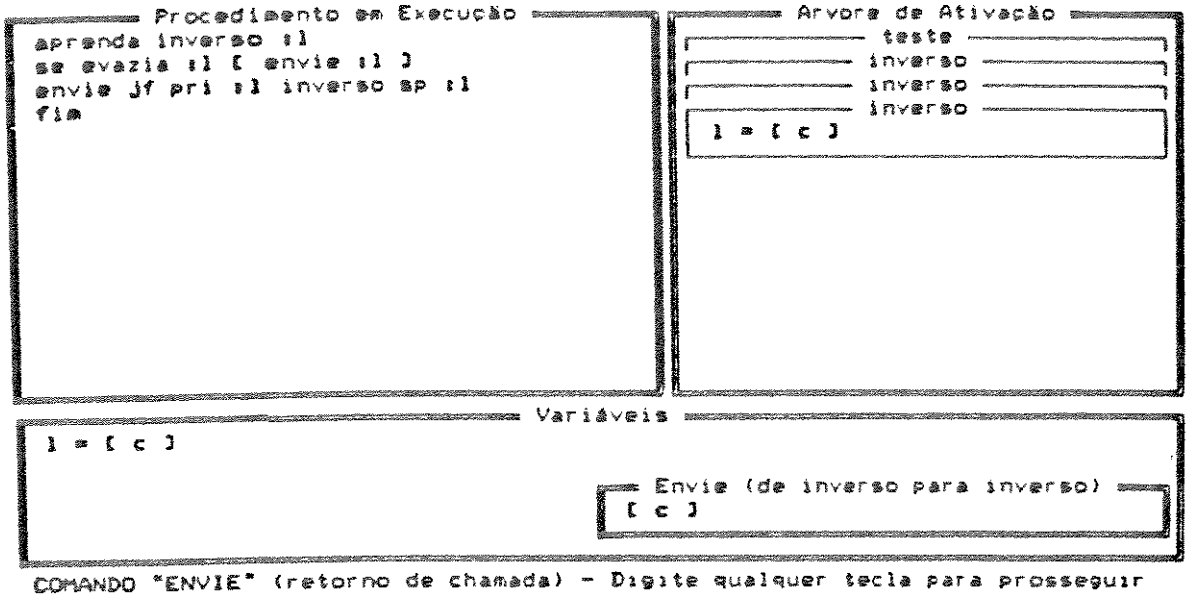


figura 32

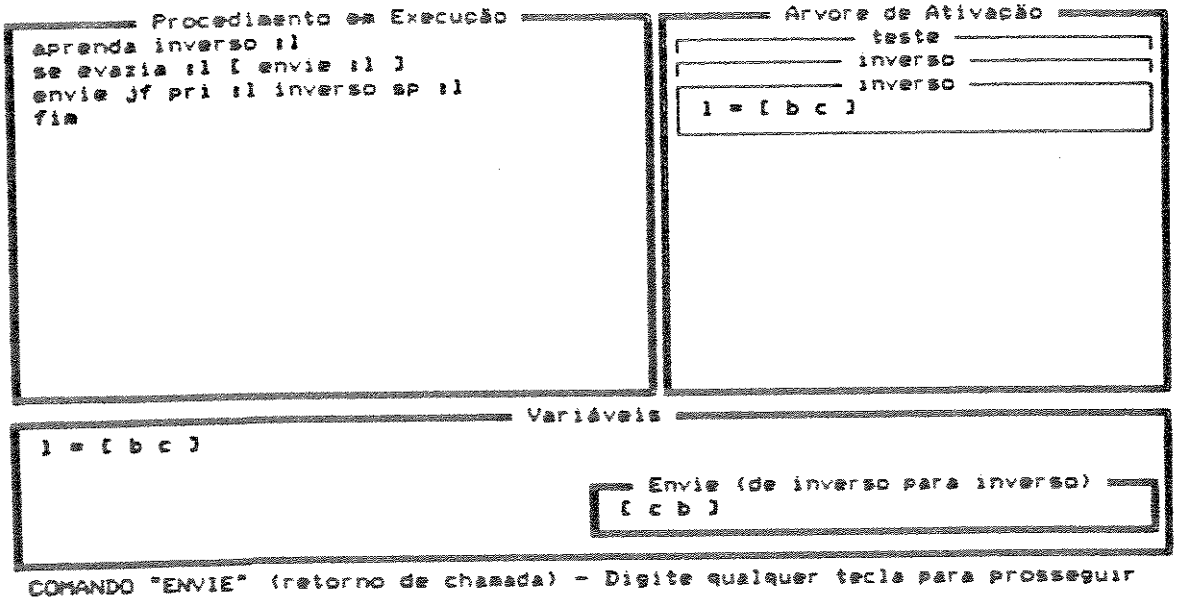


figura 33

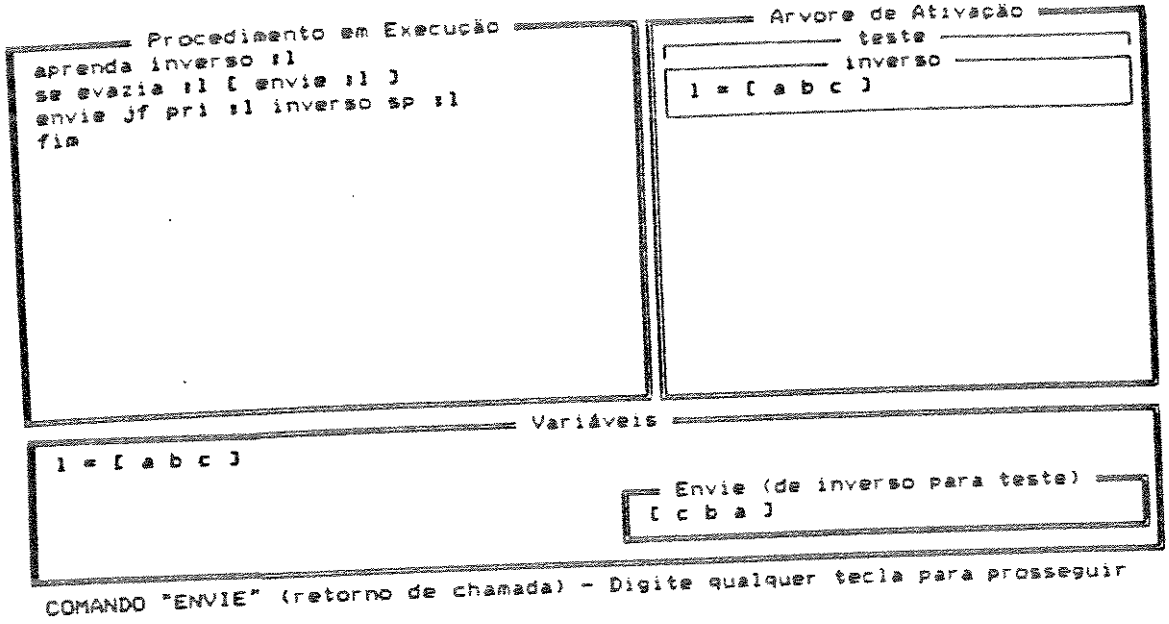


figura 34

A figura 35 mostra a execução do comando escreva no retorno do procedimento inverso e a figura 36 a tela final da execução.

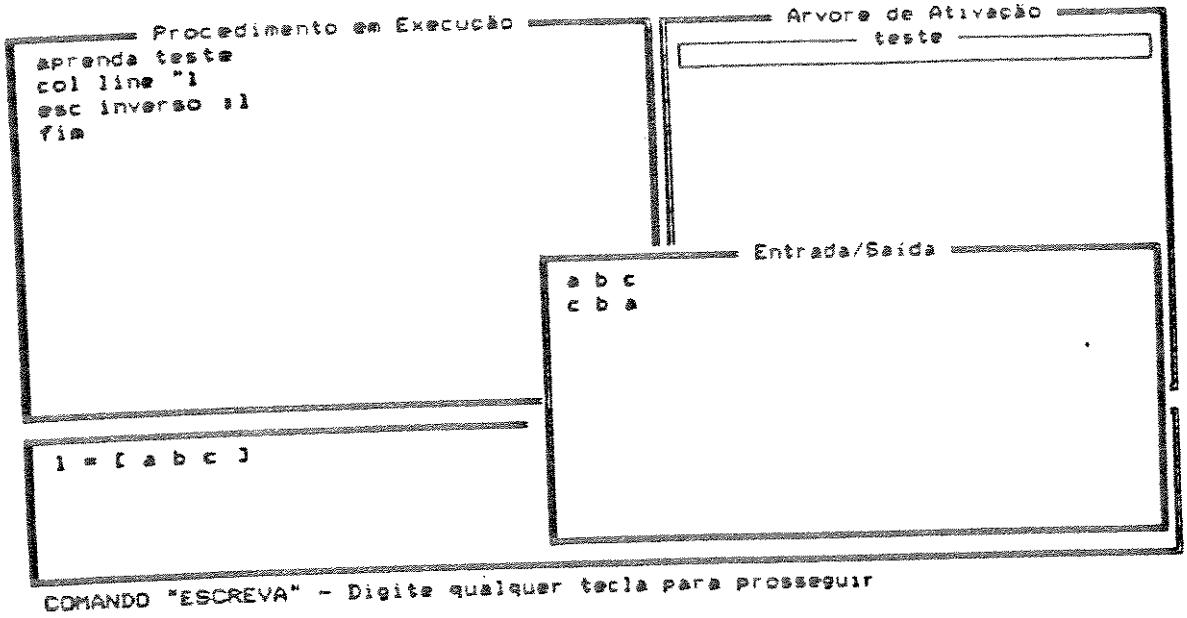


figura 35

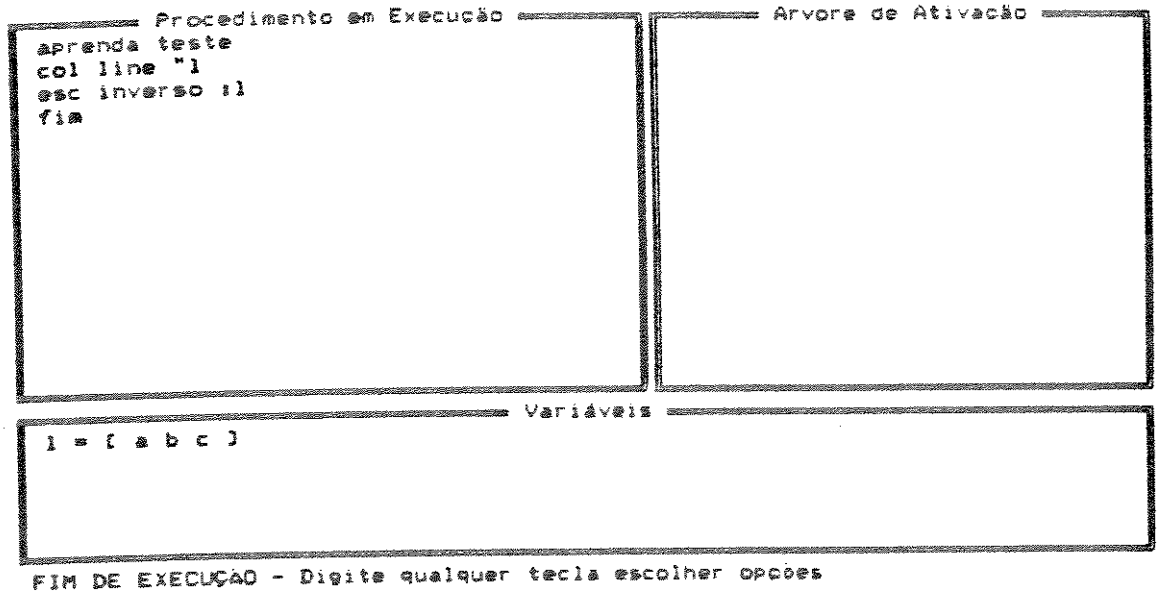


figura 36

Depois disso, pode-se escolher novas opções para reexecutar o mesmo arquivo, ou então selecionar um novo arquivo para execução, ou então, sair do sistema selecionando a opção Sair como é mostrado na figura 37.

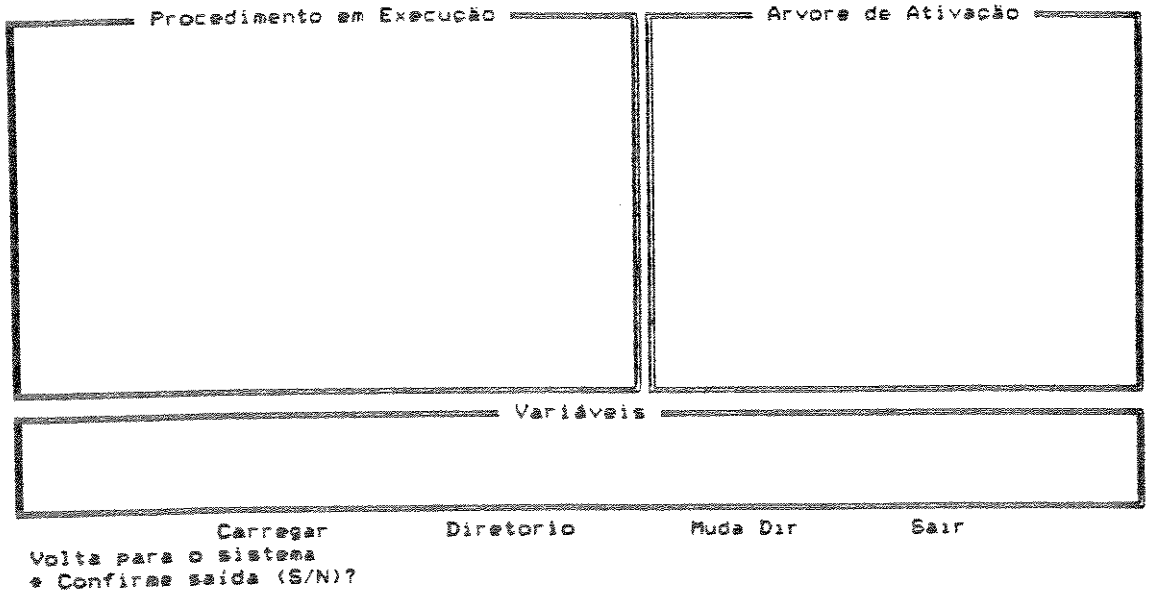


figura 37

7.4 Descrição da implementação

7.4.1 Aspectos Gerais

O sistema foi implementado em linguagem Pascal, versão Turbo 5.5. A escolha desta linguagem foi baseada em suas facilidades de tratamento gráfico e de janelas, sua portabilidade, e principalmente, na familiaridade que se tem com programação Pascal. O sistema foi implementado em um microcomputador da linha PC-XT.

Foi construído um protótipo de um interpretador para um subconjunto de comandos da linguagem Logo, onde estão incluídos comandos da parte de manipulação simbólica e comandos gerais da linguagem como comando de atribuição, repetitivo, condicional, etc.

Como já disse o objetivo não era o de obter um interpretador para a linguagem Logo e sim um ambiente de representações de um programa em execução. Neste sentido não está prevista, por exemplo, a execução de comandos primitivos no modo direto,

somente de procedimentos definidos pelo usuário e armazenados em um arquivo que é a entrada do interpretador e que corresponde a área de trabalho do sistema Logo.

Mesmo existindo estas restrições "a priori", foi feita a implementação do núcleo do interpretador de maneira a que futuras extensões pudessem ser feitas sem nenhuma dificuldade. A implementação define o sistema como um processador de listas e portanto para estender, basta que sejam previstos nós com estes novos conteúdos.

Neste capítulo relaciono o conjunto de instruções que foram implementadas e faço a descrição da estrutura de dados, pois como já disse, a estrutura do programa é de um processador de listas. E portanto descrever o programa é definir as listas que compoem a estrutura de dados.

7.4.2 Conjunto de Instruções Implementado

Foi definido um subconjunto de comandos e operações da linguagem Logo que permite praticamente todo tipo de manuseio dos objetos simbólicos da linguagem, ou seja, listas e palavras.

Todo o conjunto de instruções implementado está descrito no Apêndice.

Para permitir a construção dos objetos implementou-se as operações **JuntenoFim(jf)**, **JuntenoInício(ji)** e **Lista** que são específicas para listas e a operação **Palavra(pal)** para palavras.

Para selecionar elementos de uma lista ou palavra implementou-se as operações **Primeiro(pri)**, **SemPrimeiro(sp)**, **Ultimo(ult)** e **SemUltimo(su)** que possibilitam seleção sequencial e a operação **Elemento(elem)** para seleção aleatória.

Foram também implementados os predicados **Élista**, **Énúmero**, **Épalavra** e **Évazia**. Também implementou-se a operação **Num.Elem(nel)** que retorna o número de elementos de uma lista, facilitando a escrita de procedimentos, cujo processamento dependa do número de elementos de uma lista, de forma não recursiva.

Para efetuar entrada e saída implementou-se os comandos **Escreva(esc)** e **Ponha** para saída de valores na tela e as operações **Line** e **Care** para entrada via teclado.

Foram implementados os operadores relacionais infixos =, > e <, os préfixos Não, E, Algum e SãoIguais e os operadores aritméticos infixos +, -, * e /.

Além desses comandos e operações específicos de listas foram implementados comandos de propósito geral: Aprenda(ap), Fim, Pare e Envie para definição de procedimentos ; o comando Coloque(col) para atribuição; o comando Repita de repetição incondicional e o comando condicional em sua duas formas: Se/Então e Se/Então/Senão.

7.4.3 Estrutura de Dados

Como número de primitivas tratadas era relativamente muito pequeno não foi dispendido um maior esforço na definição da estrutura de armazenamento das primitivas. Para aumentar o conjunto de primitivas está previsto a definição de uma função de "hash" para mapeá-las. Foram armazenadas em uma árvore de busca cujo nó tem a seguinte estrutura:

nó_primitiva =

registro com os seguintes campos

- nome da primitiva;
- mnemônico;
- número de parâmetros;
- dois campos de apontadores para outro nó_primitiva

fim

Todos os símbolos definidos na área de trabalho também são armazenados em uma estrutura de árvore de busca. Observe que nesta implementação a área de trabalho é equivalente a um arquivo que é lido do disco. O nó desta árvore de átomos tem a seguinte estrutura:

nó_átomo =

registro com os seguintes campos

- nome do átomo;
- tipo do átomo que pode ser V para variável ou P para procedimento;
- Caso seja variável tem-se um campo de apontador para uma lista de dados ou para um inteiro;
- Caso seja procedimento tem-se um campo de apontador para uma lista ligada que armazena a definição do procedimento

fim

Quanto à lista de dados ela armazena uma lista ou palavra, ou então remete um apontador para a árvore de átomos.

A lista ligada que armazena a definição do procedimento, que é a estrutura básica dentro do interpretador tem a seguinte definição:

entrada_procedimento =

registro com os seguintes campos

. campo de apontador para uma lista de parâmetros;
. campo de apontador para uma lista de comandos que irá definir o corpo do procedimento

fim

Lista de parâmetros é composta de nós com a seguinte estrutura:

lista_parâmetros =

registro com os seguintes campos

. um campo de apontador para a árvore de átomos onde o parâmetro passa a existir como objeto da area de trabalho;
. um campo de apontador para o próximo parâmetro

fim

A lista de comandos que irá definir o corpo do procedimento é uma lista de apontadores para linhas do procedimento com a seguinte estrutura de definição:

```
lista_comandos =

        registro com os seguinte campos

              . um apontador de linha
              . um apontador para próximo comando

        fim
```

A estrutura de uma linha é tratada como uma sequência de elementos léxicos. São considerados elementos léxicos: primitivas, procedimento, variável precedida de ":", lista, números e simbolos especiais ([,], =, etc.). Linha tem a seguinte definição:

linha =

registro com os seguintes campos

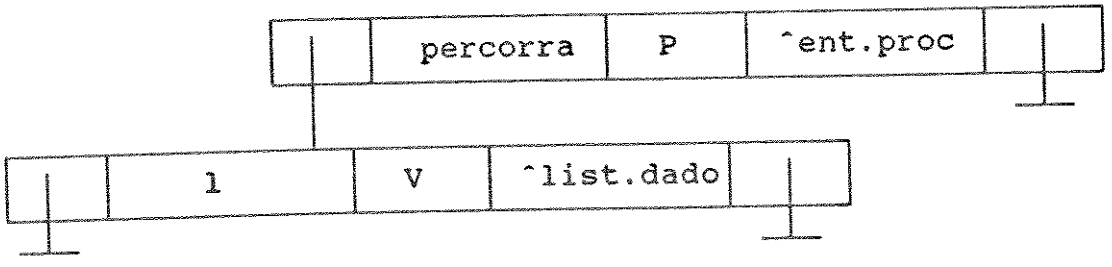
. elemento léxico que caso seja
.(PR) uma primitiva tem-se um apontador para a árvore de primitivas;
.(PO) um procedimento ou (DA) uma variável precedida de dois pontos indicando uma avaliação de valor tem-se um apontador para a árvore de átomos
.(L) uma lista, (N) número ou uma (QA) palavra tem-se um apontador para a lista de dados;
.(S) um símbolo especial, é armazenado o carácter correspondente ao símbolo;
. campo de apontador para linha (mais um elemento léxico)

fim

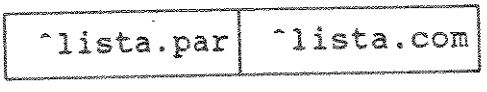
Estas são as estruturas fundamentais do interpretador. Para ilustrar como fica montada a estrutura, se tivéssemos o procedimento:

```
aprenda percorra :l
se :l = [] [pare]
percorra sp :l
fim
```

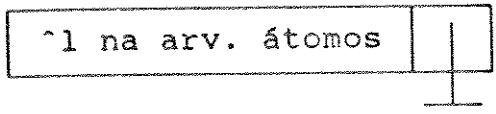
A árvore de átomos:



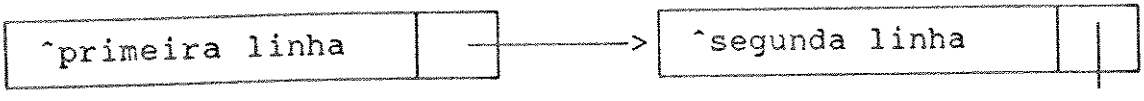
A entrada do procedimento percorra (ent.proc)



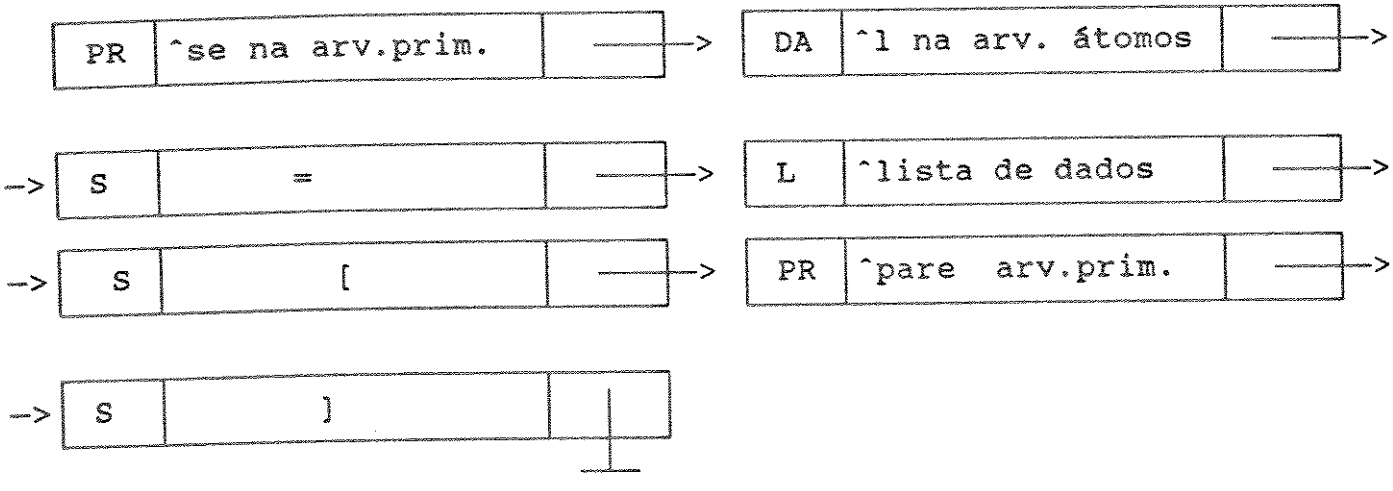
Lista de parâmetros (lista.par)



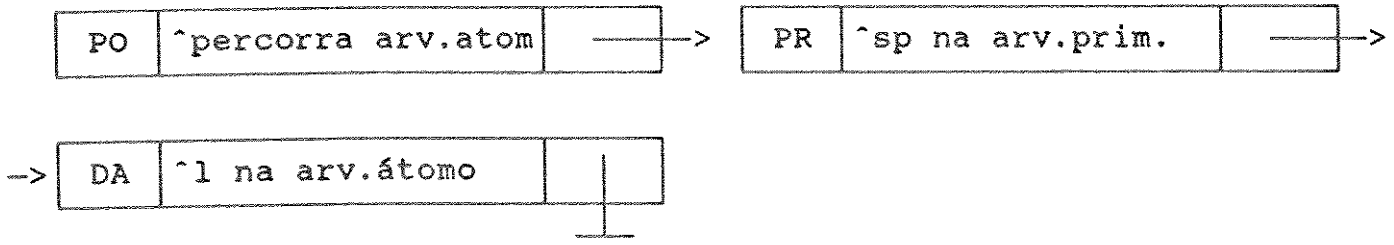
Lista de comandos (lista.com):



Primeira linha:



Segunda linha:



Portanto todo o procedimento do usuário é internalizado desta forma, que na essência pode ser vista como uma sequência de elementos léxicos.

Para a execução de um programa são utilizadas duas pilhas auxiliares, uma denominada PS que contém informações necessárias para determinar o "status" do Logo em um dado instante (nível, procedimentos chamados/chamantes, linha, etc...) e outra denominada PV que contém os valores que precisam ser armazenados para uso futuro. Estes valores são relativos à chamada de procedimentos com parâmetros. Como Logo utiliza passagem de parâmetros por valor e escopo dinâmico de variáveis, quando um

procedimento é chamado , antes de fazer as atribuições aos parâmetros formais é preciso salvar seus valores anteriores.

As representações são tratadas como chamadas de procedimentos em pontos do processamento.

A estrutura definida para o interpretador não é restrita aos comandos implementados podendo ser facilmente expandida de modo a ampliar o subconjunto de comandos implementados ou então inserir novas representações.

Capítulo 8

Uso do Sistema para Diagnosticar e Remediar o Conceito de Recursão

8.1 Introdução

O primeiro uso do sistema, foi feito durante uma oficina de trabalho que tinha como um dos objetivos oferecer um aprofundamento em programação Logo. Os participantes eram um grupo de pesquisadores e técnicos em Informática e Educação que conheciam e estavam desenvovendo atividades de programação com a linguagem Logo.

O objetivo básico deste primeiro uso era o de "depurar o uso do sistema", considerando diversos aspectos:

- aspectos associados às hipóteses iniciais sobre falhas de entendimento de conceitos de programação, aspectos esses que embasaram o desenvolvimento do sistema. Este é o ponto central do trabalho, ou seja, comprovar que as pessoas tinham falhas quanto ao entendimento de conceitos de programação e verificar em que medida o sistema facilitava o respectivo entendimento.

- aspectos associados ao próprio uso do sistema considerando-se os aspectos pragmáticos deste uso. Em suma, queria observar se a utilização do sistema não exigia grandes investimentos,

tendo em vista que o ambiente é bastante diferente dos ambientes Logo usuais.

- verificar se o sistema estava fácil de ser entendido, ou seja, se estavam adequadas, ao objetivo desejado, as formas de representação escolhidas.

Aliado a estes aspectos, pretendia depurar a forma de observação e a metodologia de uso do sistema, que havia delineado. Estavam preparadas uma série de atividades que precisavam ser depuradas, no sentido de que elas realmente atendessem aos objetivos pretendidos.

Portanto foi extremamente importante esta primeira etapa de uso e em muito substanciou outras esperiências, duas das quais descritas nos dois próximos capítulos.

Além de todos estes pontos, o grupo de pessoas envolvidas era muito interessante, pois apesar de não serem profissionais de computação, era essencial o aprendizado de programação dentro de suas atividades. Por diversas vezes, estas pessoas já haviam tentado superar a barreira gráfico-simbólico, sem sucesso.

Neste capítulo, irei descrever com bastante detalhe este uso do sistema, iniciando com uma breve descrição da oficina de trabalho em que esteve inserido.

8.2 Aspectos Gerais da Oficina de Trabalho

A oficina tinha como tema central "Representação, Reflexão e Depuração", tratando de solução de problemas nos mais diversos contextos, do Logo a Ikebana, em seguida ao Prolog, passando por LEGO-Logo e Logo Tridimensional.

A oficina como um todo era subdividida em diversas mini-oficinas. Duas mini-oficinas foram interessantes sob o contexto deste trabalho. A primeira trabalhava com recursão gráfica, onde os participantes desenvolveram procedimentos recursivos para obter desenhos extremamente sofisticados, como fractais. Esta mini-oficina teve a duração de 12 horas e de modo geral os alunos não tiveram grande dificuldade, mesmo considerando-se a sofisticação dos procedimentos recursivos que tiveram que ser desenvolvidos.

A seguir vinha a mini-oficina de programação simbólica. Esta sequência foi adotada pois pretendia-se que mini-oficina de recursão gráfica pudesse vir a ser uma boa introdução a mini-oficina de listas, tendo em vista a necessidade em dominar o conceito de recursão para desenvolver qualquer procedimento não gráfico. Foi durante as atividades da mini-oficina de listas que foi feito o primeiro uso do sistema.

A mini-oficina de listas teve a duração de 16 horas e os participantes não eram principiantes em programação. Já haviam passado por cursos de aprendizado de programação Logo e todos já programavam Logo gráfico há cerca de dois anos. Tinham um envolvimento constante com programação, e ainda não haviam conseguido desenvolver atividades não gráficas, apesar de conhecerem relativamente bem o conjunto de operações que manipulava palavras e listas.

A mini-oficina de listas era composta de uma série de atividades que visavam explorar o aspecto simbólico do Logo como contexto de representação de solução de problemas. Ao final de cada atividade era feita uma discussão onde se refletia sobre as

soluções obtidas, procurando identificar as fontes das dificuldades sentidas.

O objetivo dessas discussões era identificar quais as dificuldades que existiam ao fazer "a ponte" entre a representação da solução do problema fora do computador e a solução computacional, que deveria ser escrita na forma de um procedimento ou operação Logo. Dentro deste esquema, o uso do sistema foi inserido como uma tentativa de facilitar esta passagem, ajudando nos aspectos relativos ao entendimento de conceitos de programação.

Para mostrar como foi feito o uso do sistema e os resultados obtidos, vou relatar as fases mais significativas e que melhor atendem aos objetivos deste capítulo.

Poderá ser observado que escolhi como ponto central de observação o conceito de recursão, pois através dele se consegue detetar as outras falhas conceituais. Por exemplo, para entender recursão é essencial que se entenda controle de fluxo, passagem de parâmetros, etc.

8.3 Primeira Atividade : Diagnóstico

Para verificar o nível de entendimento de conceitos de programação que os participantes possuiam, elaborei a atividade descrita a seguir, que deveria ser feita sem o uso do computador.

A idéia de realizar atividades sem o uso do computador objetivava captar através de uma descrição escrita, escolhida pelo sujeito, características do modelo mental que ele tinha do funcionamento do procedimento em tempo de execução. Este modelo de funcionamento, a princípio, é equivalente ao modelo que o aprendiz tem do computador como máquina notacional.

8.3.1 Descrição da Atividade

"Descreva, usando qualquer representação, o funcionamento (o que faz?) de:

Obs.: o conteúdo de j é uma lista ou palavra.

A)

```
ap um :j
se évazia :j [ pare ]
esc pri :j
um sp :j
fim
```

B)

```
ap dois :j
se évazia :j [ pare ]
esc pri :j
dois sp :j
esc ult :j
fim
```

C)

```
ap três :j
se évazia :j [ pare ]
esc pri :j
três sp :j
esc pri :j
fim
```

São procedimentos de percurso de listas relativamente simples e que preservam o paradigma procedural. Os procedimentos foram apresentados um por vez, na sequência em que aparecem (um, dois e três). O procedimento seguinte somente era definido depois de feita e entregue a descrição do anterior.

8.3.2 Justificativa e Hipóteses Iniciais

O **procedimento um** usa o conceito de recursão sem volta, escrevendo os elementos de uma lista na ordem em que aparecem. Neste exercício não eram esperadas grandes dificuldades, pois como os participantes programavam Logo há um certo tempo, já utilizavam recursão e portanto deveriam ter um modelo para o funcionamento de recursão sem operações no retorno. Portanto era esperado que com as respostas a este primeiro procedimento ficasse evidenciado que os participantes tinham, pelo menos, o modelo de "loop" para recursão.

Os **procedimentos dois e três** já exigiam um entendimento mais preciso do conceito de recursão, principalmente de seu aspecto mais complexo que é o da volta da recursão.

O **procedimento dois** foi apresentado com uma particularidade que foi a de quebrar um padrão para este tipo de exercício de percurso de lista com volta que é o de "vai-e-vem", ou seja, escrever os elementos de uma palavra ou lista na ordem em que aparecem e no retorno na ordem inversa. O padrão "vai-e-vem" é apresentado no **procedimento três.**

Existiam três hipóteses sobre as respostas que apareceriam com relação ao **procedimento dois:**

- as pessoas entendiam a volta da recursão e a resposta seria correta

- as pessoas tinham já adquirido um padrão de como este tipo de procedimento funcionava, sem ter o entendimento real e então diriam que ele funciona de acordo com o padrão "vai-e-vem" só que no retorno a ordem seria a mesma que na ida. Isto porque ao invés do comando PRI está sendo utilizado o comando ULT, que no entendimento da maioria das pessoas são opostos ou inversos

- não existia realmente o entendimento do processo e as pessoas responderiam que o funcionamento era idêntico ao do

procedimento um, ignorando a existência de mais um comando ou achando que ele era inútil.

Portanto através das respostas dadas com relação ao procedimento dois é que seria definido o nível de entendimento do conceito de recursão e, associado a este, o de passagem de parâmetros.

8.3.3 Resultados Obtidos

Para o procedimento um não se pode afirmar que foram obtidas respostas erradas. Mas foi notável a dificuldade das pessoas em descrever o funcionamento do procedimento em termos do que ele faz. Em apenas duas respostas pode ser observada a visão global do funcionamento:

"Escreve o primeiro elemento da lista (ou palavra) depois o primeiro elemento da lista sem o primeiro e assim por diante. A lista vai diminuindo, perdendo o primeiro elemento até se tornar vazia. Sempre é escrito o primeiro elemento da lista."(SUJ7)

"Decompõe o objeto recebido na variável j, escrevendo cada um dos elementos numa linha, um abaixo do outro."(SUJ12)

A maioria das pessoas descreveu o código comando a comando. Isto denotou a dificuldade em visualizar o processo como um todo, sendo então efetuada uma leitura "analfabeta" do texto do programa, sem conseguir uma descrição global que sintetizasse o processo. Isto pode ser observado pela descrição a seguir:

"A atividade consta de um procedimento com variável:

- a primeira linha faz um teste se a variável, ou seja, se a lista ou palavra é vazia. se for pára o procedimento senão escreve o primeiro elemento da lista ou palavra (segunda linha).

- a terceira linha mostra que procedimento é recursivo retornando a variável (lista ou palavra) sem o primeiro elemento.

-no vídeo de um computador vamos obter a palavra ou lista em ordem decrescente."(SUJ13)

Observa-se também o ponto falho de introduzir o conceito de variável via a introdução de parâmetros. Pode ser notado que em nenhuma das descrições apareceu a palavra parâmetro ou variável

como parâmetro. A realidade é que variável em Logo, da forma como é introduzida, é parâmetro. Portanto um procedimento com parâmetro é um procedimento com variável. Poderia-se questionar se não é simplesmente uma questão de terminologia, mas na realidade esta confusão também existe a nível conceitual, como já foi exemplificado no decorrer deste trabalho. (vide Cap 4)

Estou enfatizando o aspecto das pessoas não conseguirem sintetizar o funcionamento de um procedimento, que a princípio não era meu objetivo na observação, por isto ter me surpreendido, considerando a experiência em programação dos participantes. É fundamental este aspecto de síntese tanto em programação sob o paradigma procedural quanto em programação sob o paradigma funcional. Ninguém consegue dividir um problema em partes ou então fazer composição de funções, se não tiver esta visão sintética do funcionamento de um procedimento.

Para auxiliar neste processo devem ser considerados tanto aspectos de programação relacionados a solução de problemas, bem como os aspectos relacionados ao entendimento da máquina notacional. Pois o que acontece é que enquanto está em modo direto, o usuário controla o processo de execução e quando

comandos são agrupados na forma de procedimentos, a execução foge do controle e muitas vezes o resultado é surpreendente. É a passagem do controle direto, concreto, para o abstrato. Para que isto se efetive, dentre outras coisas, é preciso ter confiança no funcionamento da máquina.

A idéia de recursão como repetição pode ser vista nas descrições:

"Primeiro há uma condição de parada do programa quando a lista é vazia. Escreve depois o primeiro elemento de j. Repete o programa escrevendo j sem o primeiro elemento até que a lista seja vazia. O programa faz escrever todos os elementos da lista." (SUJ3)

ou então,

"O interpretador vasculha uma lista j do seguinte modo:

1. verifica se é vazia se for pára, caso contrário
2. escreve seu primeiro elemento
3. repete o processo 1 e 2 com uma nova lista que é a anterior sem seu primeiro elemento."(SUJ4)

O uso de uma representação formal para descrever o processo pode ser observado a seguir:

ex:(SUJ5) Um "OLA

Nivel 0	Nivel 1	Nivel 2
ap um "OLA	se évazia "OLA [pare] esc pri :j --> O um LA -->	se évazia "LA [pare] esc pri "LA --> L um L -->

Nivel 3	Nivel 4	
se évazia "L [pare] esc pri "L -->L um "	se évazia " [pare] esc pri "	----->>res. OLA um

Este aluno foi o único que mostrou o funcionamento executando o procedimento para um exemplo. Dada a precisão da representação, talvez se justifique o fato dele estar seguro em executar o procedimento. Principiantes, de modo geral, não conseguem testar procedimentos não por não terem adquirido esta técnica de depuração, mas por não saberem como efetuá-lo. Não entendem a máquina descrita pela linguagem e consequentemente não conseguem se colocar no lugar dela.

A representação a seguir foi feita por um dos participantes que na época do curso estava programando mais em Pascal que em Logo:

"0---> define procedimento um
j---> lista
Se j é lista vazia então pára
senão escreve primeiro elemento de j
executa procedimento Um com lista j
sem primeiro elemento (recursão)
a recursão é realizada até lista j
tornar-se vazia"(SUJ10)

É notável a tentativa de obter uma representação formal do funcionamento através de um pseudo-código, muito utilizado por programadores em linguagens estruturadas. Como também é notável a insuficiência deste modo de representar, deixando a descrição extremamente confusa.

Muitos aspectos interessantes puderam ser retirados da análise das respostas aos procedimentos dois e três.

As respostas em grande parte confirmaram as hipóteses iniciais. Vamos relacionar algumas.

Os mesmos participantes que responderam de forma global e concisa, como funcionava o procedimento um persistiram corretamente na sua forma de análise, como pode ser visto nas duas descrições a seguir:

"Retira o primeiro elemento de um objeto e escreve-o e segue assim sucessivamente até esvaziar o objeto. Após isto, escreve o último elemento do objeto tantas vezes quantos forem os elementos que compõem o objeto." (SUJ12)

"Escreve a lista toda e depois escreve o último elemento dela tantas vezes quantos elementos existentes nela." (SUJ7)

Analisando a descrição dada a seguir:

"O procedimento admite uma entrada para palavra ou lista . O primeiro passo é testar se esta entrada é vazia, caso positivo interrompe a execução (seguindo o programa na quinta linha). Caso negativo, segue escrevendo o primeiro elemento da entrada.

O passo seguinte é um retorno ao programa retirando o primeiro elemento da entrada. Segue assim até atender a condição de parada. Após isso escreve os elementos da entrada na mesma ordem anterior."(SUJ1)

Está claro um dos padrões de resposta esperado, ou seja, a pessoa já foi apresentada a um procedimento deste tipo e sabe que algo é feito no retorno. A resposta até certo ponto é correta no sentido de ser executado um comando no retorno. Está claro o controle de fluxo, mas não o processo de empilhar e desempilhar o valor de variáveis locais/parâmetros. A resposta do que vai ser listado no retorno é dada em função do conhecimento do procedimento padrão que é utilizado, que é o de "vai-e-vem" já mencionado.

As respostas a seguir exemplificam um não entendimento do processo recursivo:

" ex: :j -->BOM --> B(pri) --> OM (dois sp)-->roda o prog. -->

O(pri) --> M --> roda o pg.-->M(pri) --> roda o pg -->pára

Inicia o programa com a mesma condição do programa anterior.

Se a lista for vazia, o programa pára; sem rodar o restante. Se não, deverá escrever o primeiro elemento da lista :j, chama novamente o programa sem o primeiro elemento, passa pela condição se ela ainda não for vazia, o programa irá escrever o elemento seguinte, e continuará assim até que a lista fique vazia e pare, pela condição." (SUJ9)

"O procedimento precisa de uma entrada na execução (variável).

A primeira linha faz um teste com a variável (lista ou palavra) se a mesma é vazia. Se for pára o procedimento, senão passa para a segunda linha e escreve o primeiro elemento da lista ou palavra. A terceira linha volta a executar o procedimento, retornando a variável sem o primeiro elemento (ou seja) o mesmo é recursivo até parar."(SUJ13)

Nestas descrições foi ignorado o comando após a chamada recursiva. Está claro o modelo de recursão como uma repetição condicional.

Existe também uma confusão entre o comando PARE e o comando FIM, sendo atribuido a ambos o mesmo significado. Isto é uma deficiência de entendimento de controle de fluxo que com certeza é mais difícil de ser sanada dada esta utilização na linguagem de palavras da lingua natural que, corriqueiramente, tem o mesmo significado.

Outra descrição que revela o não entendimento do processo de retorno é a dada a seguir:

"1. observa se há conteúdo em :j
2- escreve o primeiro elemento de j
3- retorna a situação inicial sem o primeiro elemento da lista
4- uma dúvida: o programa é recursivo. Portanto a última linha possivelmente será desnecessária; a não ser que sendo a lista vazia, após o pare , o esc ult :j seja executado uma vez."(SUJ2)

Interessante é o questionamento a cerca de uma instrução que nunca é executada, o que a princípio, para o participante, não fazia sentido. Aí se observa uma característica bastante comum de resposta que é dada por pessoas que possuem o modelo de repetição para recursão, mas já começam a apurar o modelo de controle de fluxo, e de alguma forma sabem que todas as instruções de um procedimento são executadas até encontrar o comando **Fim**. Mas, apesar do evidente conflito, ainda é mais forte o final da execução no **Pare** e a não existência do processo de retorno. Isto é reforçado pela resposta que o mesmo sujeito deu ao procedimento três:

"1. idem ao exercício 2, observa o conteúdo de :j

2. escreve o primeiro elemento e retorna sem o "dito cujo" para continuar escrevendo.

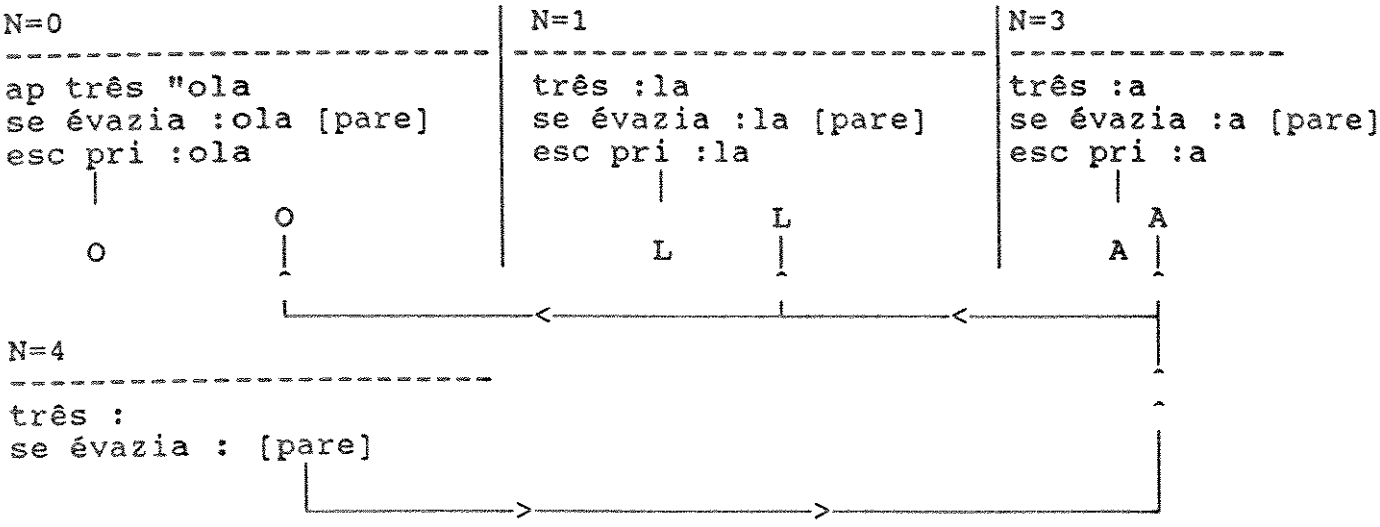
3. como o exercício anterior, a última linha é desnecessária."(SUJ2)

O sujeito (SUJ5) que fez a descrição formal do procedimento um, representando os níveis de execução, não conseguiu adequar sua representação da execução de modo a descrever o procedimento dois. Fez então uma descrição discursiva descrita a seguir:

"O interpretador Logo analisa o objeto J (palavra ou lista) entrado e verifica se possui elemento, em caso afirmativo passa a realizar a linha seguinte, isto é, escreve o primeiro elemento de J. Na quarta linha executa o procedimento com os elementos restantes de J e repete o processo anterior. Quando J não possui mais elementos o computador escreve o último elemento de J, de todos os níveis já realizados."(SUJ5)

Ele não conseguiu perceber que o último elemento em todos os níveis é sempre o mesmo. A hipótese é que ele automatizou a descrição de alguns procedimentos recursivos naquela representação, mas o entendimento ainda não foi conseguido, pois um procedimento ligeiramente diferente dos normalmente tratados em cursos e livros colocou-o em dificuldades de representar. Inclusive, em uma das tentativas de representar, ele fez uso da forma de diagrama de blocos, uma das representações mais antigas de fluxo de execução de um procedimento.

Para o procedimento três, que como já disse é muito utilizado, ele retoma a descrição formal do comportamento do procedimento. Veja a seguir:



Resultado :
O
L
A
A
L
O (SUJ5)

Resumindo em percentuais, dos treze participantes do curso apenas três demonstraram saber, ou ainda, já possuir o modelo de cópia para recursão. Isto corresponde a aproximadamente 20% dos participantes.

Questionei os participantes sobre como haviam conseguido obter resultados satisfatórios com recursão gráfica. Tinha-se grande interesse em entender como pessoas conseguem escrever um

procedimento do seguinte tipo:

```
aprenda fractal :c :n
se :n = 0 [pf :c pare]
fractal :c/3 :n-1
pe 60
fractal :c/3 :n-1
pd 120
fractal :c/3 :n-1
pe 60
fractal :c/3 :n-1
fim
```

E, apesar disto, não conseguem dizer qual o resultado produzido pelos procedimentos propostos, que utilizam recursão de uma maneira bem mais simples.

Praticamente 80% dos alunos afirmou ser esta muito mais fácil, pois bastava identificar qual era o modelo do desenho a ser produzido. Sabendo isto, bastava ir acompanhando o desenho como se estivesse sendo produzido por lápis e papel, e colocando

as operações correspondentes para a Tartaruga (pf, pd, pe, etc.).

Quando chegava em um ponto onde as formas se repetiam, era só fazer a chamada recursiva pois ali existia um "ninho de recursão". Um padrão de descrição que invariavelmente funciona, pois existe um padrão de procedimento para cada efeito a ser produzido, e o modo de ser produzido o desenho não é tão diferente do modo como o desenho seria feito a mão, com a diferença de que nos "ninhos de recursão", a "coisa" funcionava muito melhor no computador. Apenas dois, dos treze alunos, conseguiram mostrar como funcionava esta recursão. Para os demais o processo era considerado como uma "bonita mágica".

8.4 Fase de Uso do Sistema

8.4.1 Primeiro Uso

Após esta fase de descrição longe do computador, foi introduzido o sistema implementado. Depois de mostrar como é feita a interação com o sistema, o primeiro uso foi através da execução dos três procedimentos trabalhados na primeira

atividade. Este uso do sistema foi feito individualmente, por todos os alunos.

Observando a maneira como utilizaram, pude notar que a totalidade das pessoas que havia respondido incorretamente se surpreenderam quando da execução do segundo procedimento e muitas delas o executaram seguidamente diversas vezes. Um aspecto interessante é que em cada vez o foco de atenção era para uma das formas de representação e geralmente as pessoas interrompiam a execução para poder observar as representações como um todo. O modo preferido de execução foi o passo-a-passo sob comando do usuário.

A forma de uso do sistema não necessitou de muita explicação ou acompanhamento. Rapidamente os alunos conseguiram autonomia, e inclusive pude observar uns explicando para outros colegas detalhes de como utilizar o sistema.

8.4.2 Primeiros Resultados do Uso

Depois deste uso, ainda junto ao sistema foi apresentado o

seguinte procedimento:

```
ap quatro :m
se évazia :m [ pare ]
quatro sp :m
esc ult :m
esc pri :m
fim
```

O objetivo de apresentar este procedimento foi o de, utilizando ainda o paradigma procedural, explorar recursão ainda mais, porém considerando agora somente a volta. Queria verificar se o breve uso do sistema tinha conseguido algum efeito, a nível de entendimento do processo. Combinei a operação que era efetuada na volta do procedimento dois com a que era efetuada na volta do procedimento três. Não havia possibilidade de ter respostas corretas em virtude da aquisição de um padrão, pois a estrutura deste procedimento é diferente da estrutura de qualquer um dos outros três procedimentos apresentados.

Foram feitas duas perguntas, antes de efetuar a execução do procedimento:

"Considerando a chamada ? quatro "abcde

1. Vai ser escrito algum valor ?

2. Se for escrito algum valor, qual é (ou quais são)?"

Dos treze participantes todos responderam afirmativamente à primeira pergunta. Considerei um grande ganho, dado que seria esperado, caso o sistema não tivesse tido nenhuma influência, que 80% das pessoas dissesse que nada seria impresso.

Quanto a segunda pergunta:

- Sete pessoas responderam corretamente, ou seja, de que seria escrito : **e e e d e c e b e a**

- Duas pessoas reponderam que seria impresso **e e** . Estas pessoas perceberam que a execução não terminava no comando **Pare**, que tinha que ir até o encontrar o comando **Fim** e efetuaram o retorno em apenas um nível. Atingido o **Fim**, identificaram como término e não mais como retorno.

- as quatro pessoas restantes responderam que seria escrito a seguinte sequência: **e e e e e e d c b a.**

Estas últimas quatro pessoas, nitidamente combinaram a execução dos procedimentos dois e três. Fizeram o retorno em todos os níveis, mas presos ao padrão de respostas dos procedimentos já vistos.

Foi um resultado bastante significativo, dado o pouco uso do sistema. Todas as pessoas consideraram a volta, e 60% conseguiu antever o resultado correto. Foi também muito interessante observar, que muitas das pessoas antes de dar uma resposta, simularam manualmente a execução, e para isso fizeram uso de fichas de papel, "imitando" a forma como é construída a árvore de ativação no sistema. Levanto este ponto, pois com isso, um dos objetivos principais do sistema foi atingido, que é o de instrumentar as pessoas, de forma a que elas possam pensar sobre programação.

Todos os participantes observaram a execução do **procedimento Quatro**, e para os que estavam com problemas de entendimento, fiz observarem bem os aspectos relativos a árvore de ativação

que denotava os níveis de execução e que para terminar a execução era necessário que fosse efetuado o retorno em todos os níveis. Além disso fiz notar a diferença no valor dos parâmetros em cada nível.

Para certificar melhor o entendimento, solicitei que fizessem uma alteração no código do procedimento três, de modo que fosse escrita a frase "AQUI COMEÇA O RETORNO", quando fosse iniciada a escrita dos valores que eram escritos no retorno da recursão.

A proposta é diferente das anteriores, no sentido de que aqui estava sendo solicitado que fosse feita alguma coisa e não apenas o reconhecimento de um processo. Além disso, a alteração correta produziria um procedimento com estrutura diferente dos vistos anteriormente e similar à estrutura de alguns fractais, que os alunos haviam trabalhado na mini-oficina de recursão gráfica.

A alteração foi feita utilizando o sistema e as pessoas passaram bastante tempo observando novamente a execução do procedimento três e tentativas foram feitas antes de conseguir uma solução.

Todas as pessoas atingiram a solução correta, sendo que 9 na primeira tentativa.

Das quatro restantes, duas iniciaram pela seguinte solução:

```
aprenda mudaC :l
se évazia :l [pare esc [aqui começa a recursão]]
esc pri :l
mudaC sp :l
esc pri :l
fim
```

Esta solução dirigiu minha atenção para um aspecto que não havia ressaltado. De todo o trabalho feito, ficou a idéia de que o comando **Pare** não significa fim de execução e sim retorno. Mas considerando-se um dos níveis de execução, onde o comando **Pare** é executado, ele causa, como em procedimentos não recursivos, um "salto incondicional" ao comando **Fim**. Trabalhei este aspecto com estas pessoas, utilizando como exemplo, alguns procedimentos não recursivos, ou cuja recursão não era efetuada em nenhum nível. Depois disso feito, conseguiram a resposta correta.

As outras duas pessoas apresentaram de início a seguinte solução:

```
aprenda mudaC :l
esc évazia :l [pare]
esc pri :l
mudaC sp :l
esc [ AQUI COMEÇA O RETORNO]
esc pri :l
fim
```

Com estas pessoas trabalhei os aspectos relativos a sequencialidade de execução de comandos e que nada muda este aspecto. Havia uma confusão, tratando o retorno como um trecho que é executado de forma especial. Uma delas de imediato apresentou a solução correta e a outra passou pela solução discutida no caso anterior.

Ao final todos chegaram a solução correta que é:

```
aprenda mudaC :l
se évazia :l [esc [AQUI COMEÇA O RETORNO] pare ]
esc pri :l
mudaC sp :l
esc pri :l
fim
```

A partir desta fase a utilização do sistema ficou aberta a qualquer participante, em qualquer momento do curso. Todos manisfestaram desejo de utilizar somente o sistema durante o restante do curso, mas isto não pode ser feito, dado que o sistema não prevê nenhum mecanismo para tratar procedimento com erro, e portanto ainda não é adequado para ser utilizado durante desenvolvimento de um procedimento.

8.5 Mudança de Paradigma: O Procedimento como Operação

Depois desta fase inicial, que tinha como objetivo ser uma revisão de conceitos, e que propriamente não se caracterizou como revisão, comecei a trabalhar com o paradigma funcional que era o objetivo da mini-oficina.

Iniciei propondo que fossem escritos alguns predicados, que é a forma mais simples de escrever uma operação, dado que o objeto a ser retornado é uma de duas palavras, ou "VERD ou "FALSO.

Alguns dos predicados propostos, são exemplificados a seguir.

"Escreva um predicado que verifique se um número é ímpar ou não."

É um predicado bastante simples, que pode ser descrito por:

```
aprenda éimpar :num
se (sãoiguais (restode :num 2) 0) [envie "falso] [envie "verd]
fim
```

O que foi observado é que as pessoas tinham dificuldade em entender a diferença entre o comando Envie e o comando Escreva, pois aparentemente ambos produzem o mesmo resultado, ou seja, informam se o número dado como parâmetro é ou não ímpar. Também pela dificuldade em passar para outro tipo de programação, a primeira solução apresentada pela maioria das pessoas foi:

```
aprenda éimpar :num
se (sãoiguais (restode :num 2) 0) [esc "falso] [esc "verd]
fim
```

No sistema puderam verificar a diferença no funcionamento, mas para este exemplo bastante simples, não pareceu muito significativa.

Para deixar mais clara a diferença, sempre era proposta a utilização desta operação em um outro procedimento, como por exemplo:

"Dada uma lista de números, utilize o predicado éimpar definido, para escrever todos os elementos que sejam ímpares"

Isto de certa forma, levava os alunos a se certificarem da diferença, ao escrever:

```
aprenda escrevaimpar :list
se évazia :lista [ pare ]
se éímpar pri :lista [esc pri :list ]
escrevaimpar sp :list
fim
```

Mas enquanto se estava definindo predicados neste nível, persistiu o uso primeiro do comando Escreva, que depois, na resposta final era substituído pelo Envie. Isto foi adotado, como uma estratégia de solução, o que claramente denotava uma falta de segurança no novo paradigma.

Alguns outros predicados mais complexos foram desenvolvidos. Alguns exemplos são mencionados a seguir.

a) "Escreva um predicado que verifique se os elementos de uma lista são todos números ou não."

A solução já é mais complexa, envolvendo as idéias de percurso exploradas anteriormente:

```
aprenda sãonumeros :list
se évazia :list [ envie "verd ]
se não énumero pri :list [envie "falso]
envie sãonumeros sp :list
fim
```

b) "Escreva um predicado que verifique se os elementos de uma lista de números estão em ordem crescente ou não, caso a lista seja composta somente de números."

A solução deste é bastante semelhante a do problema anterior:

```
aprenda crescente :list
se évazia sp :lis [envie "verd]
se pri :list < pri sp :list  [ envie "falso ]
envie crescente sp :list
fim
```

Os problemas sempre eram apresentados com o objetivo de reforçar o entendimento de uma certa idéia ou conceito. Aqui nesta sequência de problemas estava sendo trabalhada a idéia de operação. Iniciou-se com predicados que acredito sejam as operações de natureza mais simples, principalmente na forma de uso. Depois de definir alguns predicados bastante simples, passou-se a desenvolver predicados que envolviam idéias de percurso de listas, trabalhadas nas primeiras atividades da oficina.

O que pode ser observado é que as pessoas em muito se prenderam ao paradigma procedural e enquanto trabalhavam com predicados, iniciavam o desenvolvimento do procedimento sob este paradigma e depois faziam as devidas substituições. Como é uma estratégia, que para o caso de predicados sempre funciona, os meios de quebrar ou colocar em conflito tiveram que ser postergados para outra fase da oficina. O único atenuante encontrado foi o de fazer, logo em seguida, um procedimento que utilizasse o predicado e mostrar através do uso a diferença entre a operação e o procedimento.

Quanto aos conceitos de programação, nesta fase não foram notados maiores problemas, mesmo porque a estrutura dos procedimentos era bastante simples e envolvia na maioria dos casos "recursão de cauda".

Para introduzir a idéia de que uma operação poderia retornar um objeto mais complexo como uma lista por exemplo, apresentou-se o seguinte problema:

"Escreva uma operação que retorne uma lista sem o penúltimo elemento"

Não foi notada dificuldade em escrever:

```
aprenda  sempenult :list
se évazia su :list [envie :list]
envie lista (su su :list) (ult :list)
fim
```

Mas ainda neste problema houve a larga utilização do comando escreva como primeira solução para o procedimento. Esta

estratégia só foi colocada em conflito quando foi trabalhado o problema de inverter uma lista, que discuto na próxima parte deste capitulo.

8.5.1 O problema do Inverte

Solicitei que os alunos escrevessem um procedimento que invertesse uma lista:

"Escreva um procedimento que inverta a ordem dos elementos de uma lista.

[a b c d] --------> [d c b a]"

Este é um problema bastante interessante de ser trabalhado, pois, a princípio, são esperadas três soluções, de natureza bastante diferentes.

A primeira solução esperada, e que geralmente é a mais frequente quando se apresenta este problema, é a que faz o percurso de uma lista, escrevendo seus elementos, do último ao primeiro, que pode ser descrita pelo procedimento a seguir:

```
aprenda inverte1 :list
se évazia :list [ pare ]
esc ult :list
inverte1 su :list
fim
```

Nesta solução aparece a confusão clara entre o que se vê e o que se tem. Isto é decorrência imediata do ambiente gráfico, onde o resultado de um procedimento é o que é visível na tela, um desenho. Assim, inverter uma lista é mostrá-la de forma invertida na tela.

Outra solução possível é a que utiliza uma variável para guardar a lista invertida. Esta é a solução própria do paradigma procedural e que pode ser escrita da seguinte forma:

```
aprenda inverte2 :list
se évazia :list [ pare ]
col jf ult :list :list2 "list2
inverte2 su :list
fim
```

Esta solução necessita de uma variável global **List2**, que deve ser inicializada fora do procedimento. Para fazer esta solução, o conceito de variável deve estar bem entendido, e dentro do contexto de programação Logo não é muito frequente. Geralmente esta solução é apresentada por pessoas que já programam em outra linguagem.

Além dessas, tem-se a solução funcional que pode ser expressa pelo procedimento mostrado a seguir:

```
aprenda inverte3 :list
se évazia :list [ envie :list]
envie jf pri :list inverte3 sp :list
fim
```

Esta solução exige total domínio do conceito de recursão além de muita segurança dentro do paradigma funcional de programação.

Dadas esta multiplicidade de soluções, através deste problema pode-se trabalhar muitos conceitos importantes, daí o interesse em estudá-lo.

Dos treze participantes, dois apresentaram a solução funcional, um apresentou a solução utilizando variável global e os dez restantes apresentaram a solução do percurso invertido.

O participante que apresentou a solução utilizando uma variável global, apresentou-a da seguinte forma:

```
aprenda inv :x
atr "x2 []
se évazia :x [pare]
atr "x2  ji ult :x :x2
inv1 su :x
fim
```

O problema com esta solução, é que é bastante frequente, é o da inicialização da variável dentro do procedimento recursivo. A depuração desta solução foi bastante interessante, pois enquanto estava desenvolvendo o procedimento no MSX não conseguia entender qual era o erro. Ao observar a execução no sistema, muito rapidamente ele alterou para:

```
aprenta inv :x
atr  "x2  []
inv1 :x
esc :x2
fim
```

```
aprenda inv1 :x
se évazia :x [pare]
atr "x2  ji ult :x :x2
inv1 su :x
fim
```

Foram escritos dois procedimentos, por restrição do sistema, que não aceita comandos escritos no modo direto.

Este fato ressalta a importância de se ter disponível em ambientes de programação ferramentas de auxílio à depuração.

As três soluções foram estudadas por todos os participantes e observação de quanto é diferente a execução e o resultado final foi trabalhada junto ao sistema.

Para verificar se as idéias haviam ficado realmente claras propus o problema da palíndrome, discutido a seguir.

8.5.2 O Problema da Palíndrome

Foi solicitado que utilizando o procedimento INVERTE, em qualquer uma das três versões, fosse escrito um predicado que verificasse se uma lista era palíndrome ou não. A idéia era a de inverter a lista e verificar se era igual a original ou não.

Todos os participantes imediatamente descartaram a solução do percurso afirmando que com ela "não existia lista invertida em lugar algum". Ao solicitar que explicassem melhor o sentido de lugar alguns disseram "não vai estar nem no quadrado onde fica marcando as variáveis e nem no quadrado que diz que tem valor para enviar". Foi feita uma referência à apresentação do sistema, que portanto de alguma forma estava auxiliando as pessoas a pensarem na solução.

Utilizar a operação não apresentou maiores problemas, pois foi feita uma analogia com a utilização que era feita dos predicados. A solução apresentada foi:

```
aprenda épalindr :list
se sãoiguais :list  (inverte3 :list) [envie "verd]
                                     [envie "falso]
fim
```

A discussão maior ficou em como utilizar a versão da variável auxiliar. Da discussão pude depreender que o que estava havendo era a antiga falha de entendimento do conceito de variável. Como é generalizado o conceito de que "não existe variável além de parâmetros e como estes perdem o seu valor quando termina a execução do procedimento", a dificuldade é óbvia.

Variável global é uma entidade por demais misteriosa que fica "vagando na área de trabalho". Na realidade, esta idéia de variável global na estrutura do ambiente Logo é bastante difícil. Mesmo não sendo a solução mais adequada, insisti em que fosse feita para que estas dificuldades pudessem ser trabalhadas e que com isso houvessem ganhos com relação a conceituação de

variáveis. Apesar de não ser um conceito necessário dentro do paradigma funcional, acredito ser um conceito bastante importante não só em programação como em outros contextos de solução de problemas, e que portanto deve ser bem compreendido e explorado.

Todos voltaram a verificar a execução desta versão, até se certificarem que realmente a lista invertida estava armazenada na variável X2, e que lá permanecia no término da execução dos procedimentos. Depois disso feito, simplesmente alteraram a solução da palíndrome para a seguinte forma:

```
aprenda épalindr :list
inv :list
se sãoiguais :list :x2 [envie "verd] [envie "falso]
fim
```

A maioria ao final, afirmou que esta solução era mais simples que a anterior que utilizava inverte como função. Isto talvez reforce a idéia de que realmente o paradigma procedural parece ser mais "natural" para as pessoas.

8.5.3 Atividades Finais

Utilizando esta mesma abordagem do problema do inverte e palíndrome foram trabalhados os seguintes problemas:

1. Faça um procedimento que retire um elemento qualquer de uma lista, caso ele pertença a lista.

2. Faça um procedimento que insere um elemento em uma posição qualquer de uma lista.

3. Faça um procedimento que substitui um elemento de uma lista por outro.

Como atividade final da oficina foi proposto o seguinte projeto:

PROJETO: Definir Uma Operação Para Somar Números

"Suponha que o computador é um completo ignorante em aritmética e o que se pretende é criar uma operação a qual some dois números. Não pode ser utilizada nenhuma operação aritmética do Logo. Portanto deixam de existir em Logo os operadores : +, >, <, *, /, -, resto. Uma única excessão é permitida para o operador de identidade (= ou sãoiguais). Existem dois aspectos relevantes neste projeto. O primeiro é que a operação de adição pode ser decomposta em procedimentos menores. A segunda é que números são simplesmente palavras que desempenham um papel especial dentro da linguagem. O objetivo é o de possibilitar que se pense sobre números e operações.

Este projeto gera discussões muito interessantes. É verdade que aritmética é uma parte indispensável do hardware de qualquer computador, mas na verdade o hardware é construído sobre "unidades lógicas" as quais são baseadas nas mesmas idéias que pretendemos investigar neste projeto. Como realmente o computador efetua uma soma? Está dentro do hardware? Está

construída dentro do sistema? É um circuíto? Em seu computador a adição é um circuíto? Como é a adição para crianças? É uma capacidade que está embutida ou são peças de conhecimento que são adquiridas?

O difícil é que se está tão familiarizado com adição que não se consegue mais identificar seus componentes.

O ponto de partida do projeto é que o computador está totalmente destituído de qualquer aritmética. Como ensiná-lo a somar? Por onde iniciar? O que se pretende é ensinar o computador, será que podemos fazer uma analogia com o processo de ensinar uma criança a somar?

Neste ponto duas sugestões podem aparecer. Professores vão sugerir que se deve "ensinar fatos sobre números" e profissionais de computação provavelmente vão sugerir que se construa uma tabela de somas 10x10. Destas sugestões surgem questões óbvias. Aos professores pode-se questionar como ensinar "fatos sobre números, quais são eles e quantos são necessários". Aos "computeiros" questiona-se se uma tabela 10x10 é o

suficiente, e como deve ser construída e organizada. Fazer uma tabela é um modo de ensinar fatos sobre números?

Que espécie de tabela e quais são os fatos sobre números. Uma tabela com a soma dos 100 primeiros números é muito limitada e mesmo que se construa uma tabela maior ela vai continuar sendo limitada. É isso que as pessoas tem na cabeça? Será que não existe uma ou mais idéias chave que podem ser construídas sem que com isso se esgote a memória do computador?

Esta questão é análoga a pensarmos se uma criança aprende que 14+20=34 como uma noção primitiva ou existe uma idéia básica fundamental que norteia este aprendizado? O que as crianças aprendem sobre números? Elas aprendem a relação entre eles. Elas aprendem a ordená-los. Aprendem a reconhecer os dígitos e sua ordem. Elas aprendem que UM é o nome de 1 e ONZE é o nome de 11 e que CENTO E ONZE é o nome de 111. Elas aprendem que 11 é diferente de 2. Podemos pensar nisto de forma diferente. Digamos que 1 é uma palavra especial. Pode-se criar uma nova palavra pondo-a junto com outra. Portanto PAL 1 1 é 11 ou ONZE. Concatenar é uma forma de modificar números.

Vamos retornar ao aprendizado relativo a reconhecer e ordenar dígitos. Isto precisa ser ensinado ao computador. Podemos dizer que se vai ensinar o computador a contar. Pense sobre o que está envolvido no processo de contar. Quantos símbolos estão envolvidos? O que voce quer é ensinar ao computador que 7 vem depois do 6, que 10 vem depois do 9 e assim por diante. Vai ser preciso definir algumas regras que especifiquem o que fazer para produzir o próximo número em sequência. Isto é o que voce vai precisar fazer. Construa seu plano e implemente-o.

Ao completar esta parte do projeto voce terá descrito a operação de somar 1. Será que sabendo somar 1 se pode somar quaisquer dois números?

Será que este trabalho feito não pode ser utilizado para construir a operação de subtração?

Provavelmente voce construiu seu projeto pensando na soma e subtração de números positivos. Voce pode extendê-lo para que possa tratar também números negativos.

Agora que voce contruiu uma primitiva tão primitiva, porque não fazer o mesmo com as outras operações aritméticas?"

Os resultados obtidos foram bastante satisfatórios e o sistema foi amplamente utilizado como um auxilio a depuração, mais a depuração de idéias pois ele não trata procedimentos com erro.

8.6 Conclusão

Este primeiro uso foi extremamente gratificante e apresentou mais resultados que o esperado.

A utilização do sistema foi facilmente aprendida, o que demonstrou ser um ambiente amigável.

As representações se mostraram adequadas no sentido de que foi notado que os participantes se apoderaram delas ao fazerem uso de representações semelhantes as do sistema, mesmo longe do sistema. E como o sistema a princípio não tem como objetivo ensinar e sim instrumentar de modo a facilitar o aprendizado,

esta foi a observação mais significativa quanto a consequência do uso.

Quanto as falhas de entendimento conceitual, pode-se resumir em alguns pontos:

- **a terminologia da linguagem de programação pode complicar o entendimento do funcionamento:** um dos problemas que é gerado pela utilização de termos da lingua natural em linguagens de programação é a ambiguidade da lingua natural. E como para principiantes o que se passa dentro do computador é um mistério, existe a tendência de julgar que o entendimento da máquina é muito maior do que é na realidade, sendo semelhante ao nosso nível de entendimento. O papel de uma linguagem de programação, mais especificamente do Logo, como meio de expressão de processos mentais, é um tópico a ser explorado em futuras pesquisas.

- **O perigo da aquisição de padrões de procedimentos que funcionam.** É importante ressaltar um fato que foi muito comum durante esta minha fase de observação das pessoas trabalhando com Logo: o de se adquirir certos padrões de procedimentos e de

seu repectivo funcionamento. Muitas vezes respostas corretas não significam um entendimento e sim que se está questionando sobre um mesmo tema, sem nenhuma novidade que coloque em conflito um padrão de funcionamento já adquirido. Esta é uma das principais falhas de avaliações que são efetuadas que geralmente solicitam somente um pouco mais do mesmo, e daí uma resposta com sucesso não ter grande significado.

- O trabalho no ambiente gráfico não favorece o entendimento de como o computador efetivamente processa um procedimento. O ambiente gráfico, mesmo quando se considera o mesmo paradigma procedural, permite a obtenção de resultados bastante interessantes mesmo que não se saiba direito o que está acontecendo quando o programa está funcionando. O que acontece é o que é visível, ou seja, o desenho na tela. No processamento simbólico, o que acontece a cada passo não pode ser acompanhado e nem imaginado, pois o processo que qualquer pessoa faria para produzir os resultados desejados seria muito diferente do que é adotado na máquina.

- para entender o conceito de recursão é necessário entender como o computador processa um procedimento recursivo: as pessoas precisam ser instrumentadas para poder pensar sobre procedimentos recursivos e consequentemente entender o processo. Como é um processo abstrato e não visível é indispensável a utilização de representações do processo recursivo. Neste sentido o sistema mostrou-se bastante adequado, pois em muitas ocasiões pude observar as pessoas fazerem uso de representações semelhantes as que são apresentadas no sistema para poder acompanhar um procedimento durante um processo de depuração. Pode ser observado que as pessoas conseguiram uma melhor estruturação do domínio de programação, o que reflete o aperfeiçoamento de um modelo mental do funcionamento da linguagem.

- para entender o conceito de variável em seu sentido mais amplo é necessário dissociá-lo do conceito de parâmetros: esta é uma falha séria gerada do aprendizado de Logo. Por ser uma linguagem de bases funcionais torna-se natural a introdução de variáveis como parâmetros. Mas como existe o ambiente procedural, este conceito deve ser estendido para o conceito amplo de variáveis. Neste aspecto o sistema também mostrou-se adequado, pois durante a apresentação dinâmica da execução,

ficava clara a diferença entre parâmetros que existiam durante a execução de um procedimento e as variáveis globais que persistiam, com seus valores alterados ou não durante a execução dos procedimentos, e que podiam ser utilizadas livremente, como parâmetros ou não.

- a mudança de paradigma está mais associada aos aspectos computacionais de programação que aos aspectos de solução de problemas: mudar de paradigma implica em mudar o contexto de representação da solução de um problema. Pelo que pude observar nesta oficina, a natureza da solução do problema não é profundamente alterada. A distância entre a solução e sua representação na forma de um procedimento é a mesma considerando-se o paradigma procedural ou funcional. A segurança da pessoa com relação a como a máquina irá resolver o procedimento escrito é que é crucial. E esta segurança só é conseguida a medida que o processo de execução é entendido. E para este entendimento a visualização do processo de execução é fundamental, fornecendo uma resposta que possibilite adicionar novas componentes ao modelo mental do comportamento da linguagem.

Capítulo 9

Uso do Sistema no Aprendizado de Recursão no Processamento Simbólico

9.1 Introdução

Um uso mais prolongado do sistema foi feito em um curso de 40 horas, oferecido para um grupo de professores da rede pública de ensino participantes de um projeto de introdução de computadores em escolas, utilizando programação Logo.

O trabalho com este grupo foi um grande desafio. Já vinha trabalhando com estes professores há três anos e eles sempre ofereciam grande resistência em se aprofundar em programação. Já tinha feito algumas tentativas e sempre com um rendimento bastante baixo.

Apesar da necessidade de um aprofundamento para poder conduzir o projeto que estavam envolvidos, achavam a parte de manipulação simbólica muito difícil e muito diferente do Logo que conheciam.

Dentro desta perspectiva eu tinha como ponto central de observação o quanto o sistema iria viabilizar este aprofundamento junto a este tipo de usuário, que necessitava de programação mas não podia ser qualificado de especialista. Minha hipótese era de

que se fossem conseguidos resultados satisfatórios em muito isto seria creditado ao uso do sistema, pois outras tentativas já haviam sido feitas.

Também participaram deste curso um pós-graduando em Pedagogia envolvido com o uso de Logo em Educação e um engenheiro elétrico envolvido com o desenvolvimento de periféricos e interfaces computacionais para educação.

Portanto o grupo como um todo era composto por 6 pessoas envolvidas com projetos de Informática e Educação, assim divididos:

- 4 professores da rede pública, a nível de segundo grau. Destes quatro professores, dois eram da área de Matemática, um de Física e Química e um de Português

- 2 técnicos do Núcleo de Informática Aplicada à Educação da UNICAMP, sendo um engenheiro elétrico e outro mestrando em Pedagogia.

Todas as pessoas tinham o mesmo tempo, cerca de 5 anos, de programação em Logo e, "a priori", os mesmos conhecimentos, ou seja, um certo domínio do ambiente gráfico. Digo um certo domínio, pois eles nunca tinham desenvolvido projetos gráficos sofisticados, como o desenho de fractais. Não tinham qualquer familiaridade com as operações da parte de listas, apesar de já as terem visto em outras tentativas de introdução que haviam sido feitas.

A metodologia de trabalho foi semelhante a desenvolvida com o grupo descrito no capítulo anterior. Foram definidas atividades que de alguma forma conduziriam a uma ampla exploração do ambiente de manipulação simbólica. Como o curso era de maior duração, também foi dada muita ênfase a fase de solução do problema, ressaltando os tipos de problemas que podiam ser resolvidos no contexto simbólico.

Para atender aos objetivos de minha pesquisa, que corriam em paralelo aos objetivos específicos do curso, programei algumas atividades específicas de diagnóstico. Quanto ao uso do sistema, nada em especial foi programado, ficando aberto ao uso durante

todo o curso. Mas sempre que faziam uso do sistema eu observava os pontos falhos e apresentava um pequeno exercício de remediação.

Como os participantes não conheciam as operações que manipulavam palavras e listas, iniciei a primeira fase de diagnóstico fazendo uma atividade utilizando o Logo gráfico que lhes era familiar.

Depois destas primeiras atividades de diagnóstico, apresentei o sistema e as operações de manipulação simbólica. Em seguida, passei por outra atividade de diagnóstico, agora no universo simbólico. Com isso procurei cobrir as deficiências da primeira fase de diagnóstico e introduzir o uso efetivo do sistema. Foi utilizada a mesma sequência de atividades feita com o grupo cujo trabalho está descrito no capítulo anterior.

Durante o desenrolar do curso o sistema foi amplamente utilizado. E no final, fiz mais uma bateria de atividades que também visavam um diagnóstico das mudanças ocorridas, quanto ao entendimento dos conceitos computacionais.

A seguir, vou detalhar as fases mais significativas para este trabalho. Novamente vou enfocar toda minha análise no conceito de recursão.

9.2 Primeira Fase de Diagnóstico

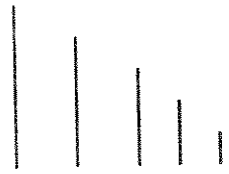
Este diagnóstico inicial foi feito dentro do ambiente gráfico, pois os alunos não conheciam as operações de manipulação simbólica. Foram atividades descritivas, que objetivavam verificar o modelo que as pessoas tinham do funcionamento de um código.

Solicitei que descrevessem, sem utilizar o computador, o funcionamento dos seguintes procedimentos:

a)
```
ap um :lado
se :lado < 0 [pare]
ação1
um :lado - 10
fim
```

```
ap ação1
pf :lado pt :lado
un pd 90 pf 20 pe 90 ul
fim
```

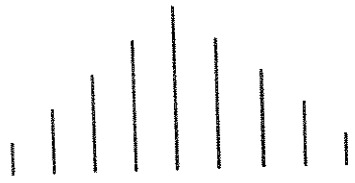
Estes procedimentos produzem um desenho do seguinte tipo:



b)
```
ap dois :lado
se :lado > 70 [pare]
ação1
fim

ap ação1
pf :lado pt :lado
un pd 90 pf 20 pe 90 ul
fim
```

O desenho produzido é do seguinte tipo:

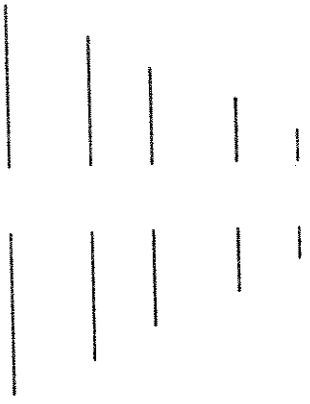


c)
```
ap quatro :lado
se :lado < 0 [ação3 pare]
ação1
quatro :lado -10
ação1
fim

ap ação1
pf :lado pt :lado
un pd 90 pf 20 pe 90 ul
fim

ap ação3
pd 180
un pd 90 pf 20 pe 90 pf 20 ul
fim
```

Desenho do seguinte tipo:



Os problemas iam em um crescente de complexidade, considerando o caso mais simples de recursão até um mais sofisticado que tem uma ação antes de iniciar o retorno. Foram apresentados um em seguida ao outro. Depois que era dada uma resposta o problema seguinte era proposto.

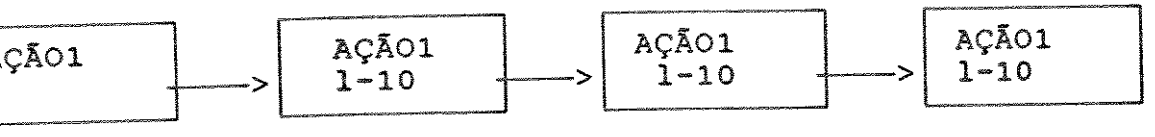
As respostas obtidas com relação ao primeiro problema foram todas corretas. Todos os participantes acompanharam a execução do procedimento como se fossem a Tartaruga e conseguiram depreender o que fazia. Algumas respostas ilustram o padrão de respostas:

"Traços verticais de tamanhos diminuídos de 10 até o zero e com espaços de 20 entre eles" (SUJ2)

"O procedimento faz linhas paralelas e decrescentes em 10 até < que zero, e com espaços constantes entre elas" (SUJ3)

Algumas pessoas representaram o fluxo de execução, de forma similar as que são mostradas a seguir:

UM ----> Ação1 ----> UM ----> Ação1 ----> UM (SUJ3)



(SUJ5)

<zero

pare

São formas bem pessoais de representação, não sendo padrão de nenhum tipo de curso. Na segunda forma de representar o fluxo, somente é representado o procedimento que efetivamente faz alguma coisa, ignorando a função de controle do procedimento UM. Este é um problema de entendimento do fluxo de controle que também pode ser observado em outras respostas.

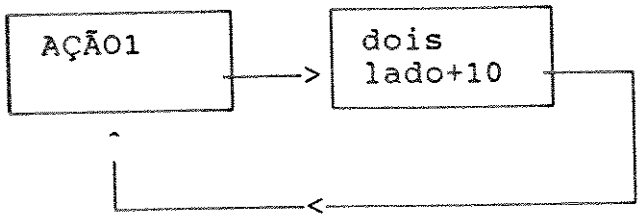
Com relação ao segundo problema, onde já havia operação a ser efetuada no retorno, notou-se uma maior dificuldade em simular a execução, se colocando no lugar da Tartaruga.

Somente duas pessoas responderam corretamente. As demais afirmaram que seriam construídas linhas paralelas, num crescente de 10, até que o tamanho se tornasse 70. Ou seja, apresentaram o seguinte esquema de solução:

Ignoraram o retorno da recursão. Como desconheciam como isto é feito, não mais conseguiram se colocar no lugar da Tartaruga.

Foram também significativas as tentativas de representar graficamente o esquema geral da execução. Mas as tentativas não conseguiam expressar o retorno. Alguns exemplos são:

dois 60 ---> Ação1 60 ---> dois 70 ---> Ação1 70 --->dois 80
(SUJ3)



até chegar no 70 pare (SUJ5)

Com relação ao terceiro problema, nenhuma resposta foi correta. Dos seis participantes, quatro não conseguiram chegar à uma resposta. O desentendimento foi generalizado e das duas pessoas que responderam, uma deu a seguinte resposta:

"Iniciei com Ação1, pf 40, pt 40, un, pd 90, pf 20, pe 90, ul, utilizei os comandos da Ação3, voltei para Ação1, tirando 10 até chegar em zero"

Esta descrição, que nada mais é que uma leitura do código, com a pessoa se colocando no lugar do computador, demonstra que o desconhecimento de como a ação é efetuada leva as pessoas a não terem mais o poder de sintetizar a função de um procedimento.

Quando este entendimento era claro, no caso do primeiro procedimento, não se obteve nenhuma resposta deste tipo.

De modo geral, apesar de poder observar alguns aspectos interessantes, não julguei esta atividade adequada. Foi muito difícil para as pessoas obterem uma resposta e a princípio não ficou claro se a dificuldade se atinha apenas ao não entendimento de aspectos conceituais. Muitas vezes, acompanhar uma execução gráfica é muito difícil, e de modo geral é uma atividade tediosa, o que prejudica os resultados. Acredito que melhores resultados teriam sido obtidos, se ao invés de solicitar que descrevessem o funcionamento, tivesse sido proposta a feitura dos procedimentos.

Após efetuada estas descrições, as pessoas executaram os procedimentos no MSX e os resultados foram discutidos. Não pode ser utilizado o sistema, porque ele não possui implementadas as características gráficas do Logo.

Em conclusão, pode-se a princípio estabelecer que 50% dos alunos tinham já uma idéia mais elaborada de como funcionava a recursão, estando bastante próximos do modelo de cópia. As outras

50% tinham claramente o modelo de repetição. Todas tinham dificuldades com relação ao fluxo de controle quando da chamada e retorno de procedimentos, bem como do processo de restauração do valor de parâmetros no retorno da recursão.

Depois desta atividade foram introduzidas as operações da parte de manipulação de listas. Foram desenvolvidos exemplos bastante simples, muitos em modo direto e procedimentos sem recursão. Foram ressaltadas as diferenças de uso de procedimentos que eram operações dos procedimentos que expressavam comandos.

Fiz a introdução do uso do sistema bastante cedo, utilizando-o para auxiliar a entender o funcionamento das operações.

9.3 Introdução ao Uso do Sistema

O objetivo deste primeiro uso era apresentar o sistema e possibilitar o uso de modo a obter familiaridade com o ambiente.

Observaram a execução de procedimentos previamente definidos, explorando os aspectos de manipulação de todas as possibilidades do sistema. Os procedimentos eram do seguinte tipo:

```
aprenda teste
col 1 "a
col 2 "b
teste1 5
teste1 :b
esc lista :a :b
fim
```

```
aprenda teste1 :b
col 5 "a
col 6 "b
esc lista :a :b
teste2 :b
esc lista :a :b
fim
```

```
aprenda teste2 :a
col 8 "a
col 9 "b
teste3 :a
esc lista :a :b
fim
```

```
aprenda teste3 :a
col 11 "a
esc lista :a :b
fim
```

Aproveitei para enfatizar aspectos simples de fluxo de controle, parâmetros e variáveis, fazendo-os observar as representações. Além disso, puderam explorar o funcionamento das operações com palavras e listas.

De modo geral, se mostraram bastante interessados e não mostraram dificuldades em utilizar o sistema. Um dos participantes definiu o sistema como sendo **"o Raio-X do procedimento"**.

Depois deste primeiro uso e de certificar o entendimento inicial do funcionamento das operações que manipulavam listas e palavras, desenvolvi uma segunda atividade de diagnóstico. O objetivo, foi tentar clarificar os resultados obtidos no primeiro diagnóstico, que não julguei muito adequado.

Para isso utilizei as atividades que solicitavam identificar o resultado de procedimentos que faziam o percurso de listas.

9.4 Segundo Fase de Diagnóstico

A seguir são apresentados os problemas e a análise de algumas respostas.

9.4.1 Descrição dos Problemas

Utilizando qualquer representação, diga o que faz:

Atividade 5

```
ap um :k
se évazia :k [ pare ]
esc pri :k
um sp :k
fim
```

Atividade 6

```
ap dois :j
se évazia :j [ pare ]
esc pri :j
dois sp :j
esc ult :j
fim
```

Atividade 7

```
ap três :l
se évazia :l [ pare ]
esc pri :l
três sp :l
esc pri :l
fim
```

Atividade 8:

```
aprenda quatro :m
se évazia :m [ pare]
esc ult :m
quatro su :m
esc pri :m
fim
```

9.4.2 Resultados Obtidos

Não foram obtidos resultados muito diferentes dos mencionados no capítulo anterior.

Novamente foi observada a aquisição de padrões de funcionamento de procedimentos. O nível de acerto para o típico procedimento de percurso de lista que utiliza volta de recursão, que é o procedimento três, foi mais de 80% Enquanto para procedimentos bastante similares, como os procedimentos dois e quatro somente uma pessoa respondeu corretamente. Então obtinha-se respostas do seguinte tipo, para um mesmo participante:

ativ6: escreve os elementos de uma lista ou palavra e depois escreve o último elemento e pára

ativ7: escreve os elementos de uma lista ou palavra e depois escreve de trás para frente

ativ8: escreve os elementos da lista ou palavra de trás para frente e escreve o primeiro da lista e pára.

Pode-se perceber claramente que o participante não entende a volta da recursão, pelas respostas que são dadas às atividades 6 e 8. A resposta correta à atividade 7, perde o significado, ficando evidenciado que o participante já conhecia este

procedimento e conhecia seu resultado. O participante sabe o que o procedimento faz, mas não sabe como faz.

Como já mencionei, apenas um participante respondeu corretamente a todos os exercícios, fornecendo respostas do seguinte nível:

ativ6: escreve cada elemento da lista do primeiro ao último. Depois da parada escreve o último elemento um número de vezes igual ao número de elementos da lista

```
ex: dois [a b c d]
    a
    b
    c
    d
--------
    d
    d
    d
    d
```

ativ7: escreve cada elemento da lista do primeiro ao último. Depois da parada escreve os elementos da lista do útimo para o primeiro

```
ex: três [a b c d]
    a
    b
    c
    d
   -------
    d
    c
    b
    a
```

ativ8: escreve os elementos da lista de trás para frente, do último até o primeiro. Depois escreve o primeiro elemento tantas vezes quantas forem o número de elementos da lista.

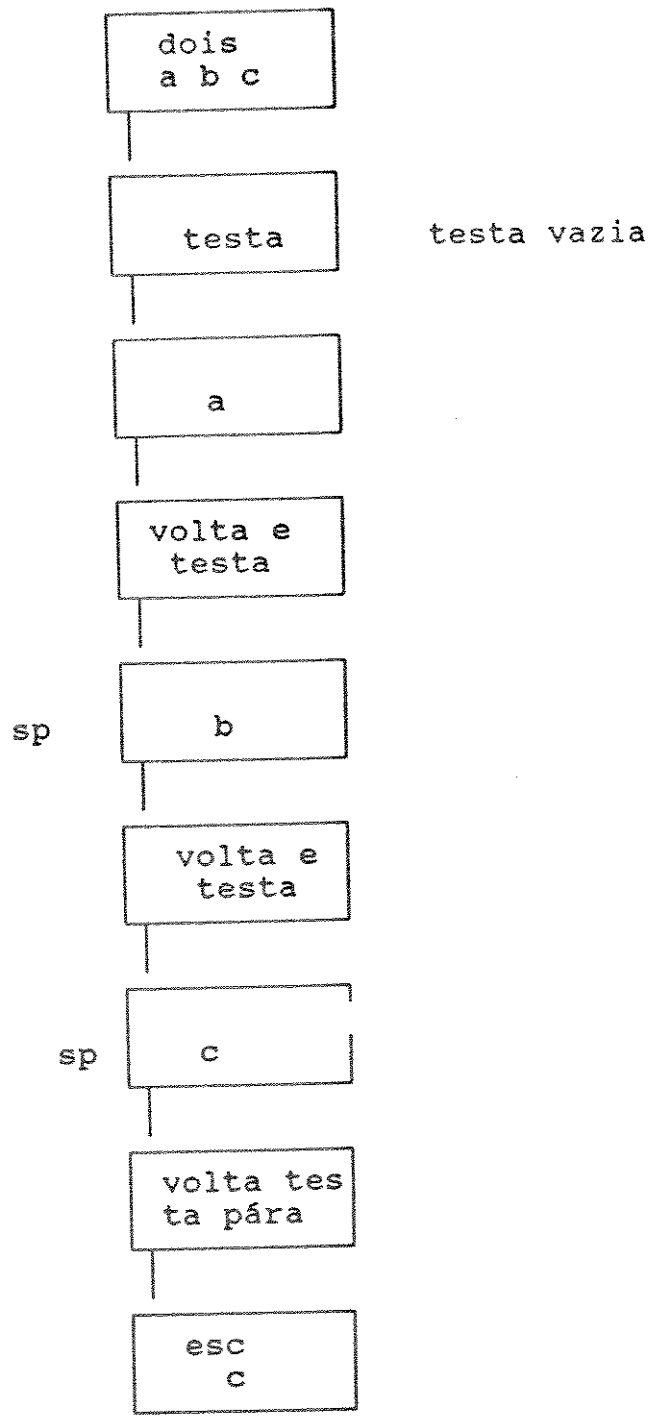
```
ex : quatro [a b c d]
     d
     c
     b
     a
-------------
     a
     a
     a
     a
```

Foi o único participante a representar a solução exemplificando, ou seja, acompanhando para um valor de entrada, o que demonstra segurança quanto ao processo de execução. Isto para mim evidencia o aspecto executável de modelos mentais. Este participante já possuia um bem elaborado modelo mental de

processos recursivos, e consequentemente este modelo é funcional podendo ser utilizado com segurança para prever resultados do comportamento do procedimento.

As pessoas com maior dificuldade inicial em programação foram as que a princípio demonstraram uma maior correlação das representações que utilizaram para responder aos exercícios e as representações do sistema. Algumas formas gráficas foram apresentadas, como as exemplificadas a seguir:

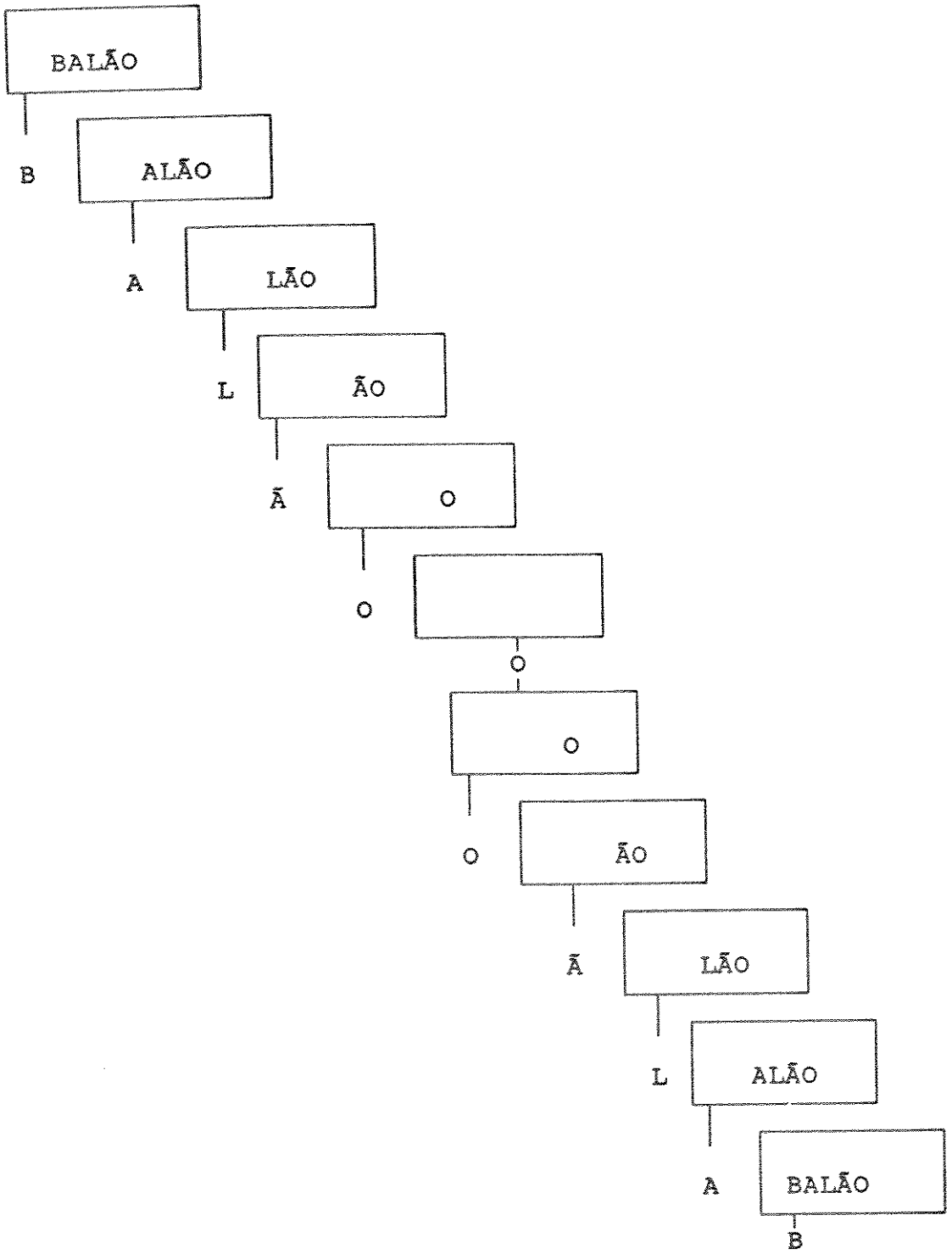
ativ6:


dois
a b c
testa
testa vazia
a
volta e
testa
sp
b
volta e
testa
sp
c
volta tes
ta pára
esc
c

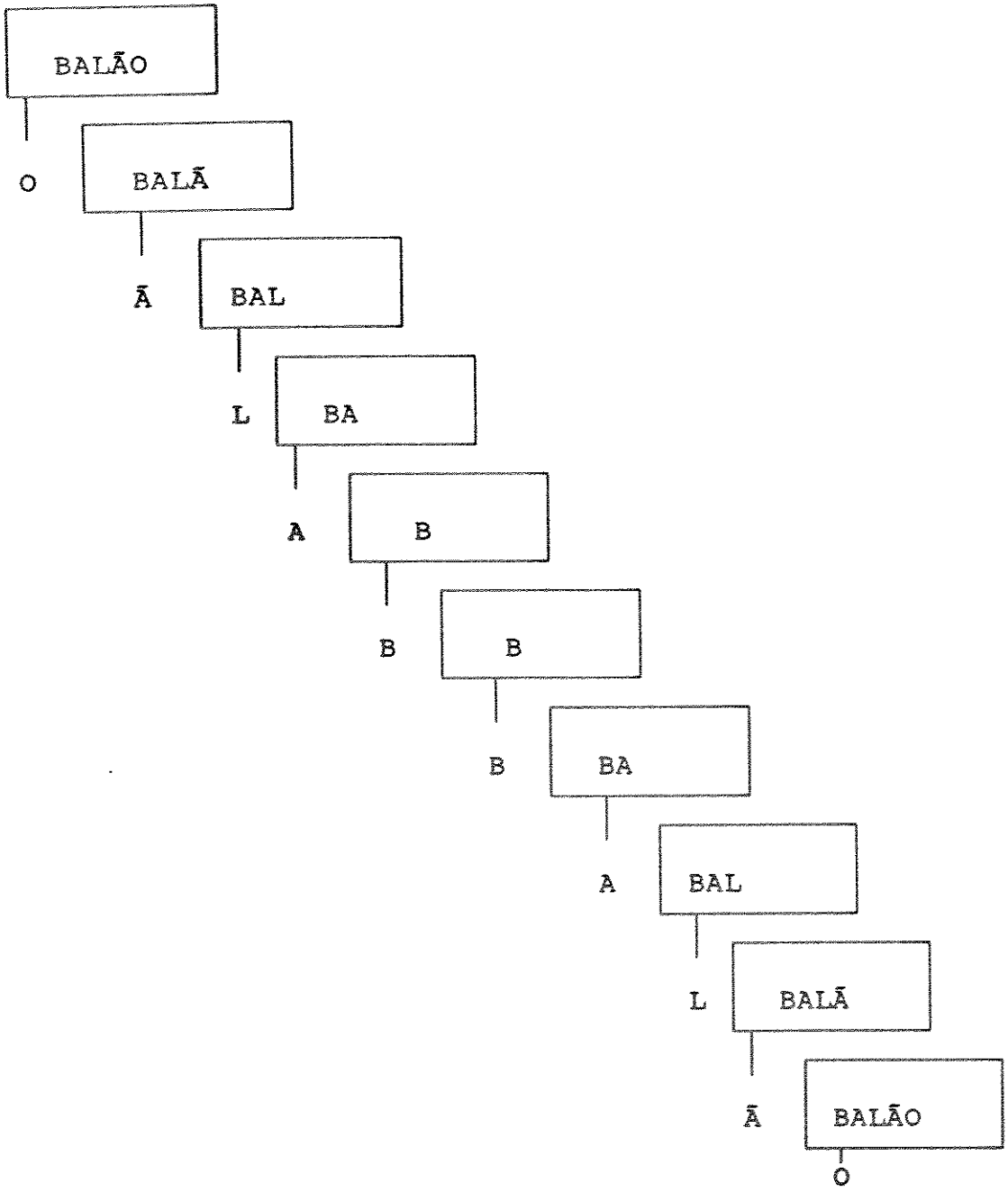
Neste caso não foi representado em nenhum caso o retorno. Toda execução foi feita até encontrar o comando Fim, mas não considerou os níveis de retorno. Um esquema de fluxos em caixas, onde pode ser observada uma mistura sobre a função das caixas. Apesar de já estar de certa forma tentando estruturar uma forma de representação, este esquema de caixas em muito dificulta a representação do retorno.

Outra forma gráfica de representação utilizada por um dos participantes pode ser exemplificada pelas duas respostas a seguir:

ativ7: quando entro com uma palavra ou lista, ele primeiro verifica se se ela é vazia, como não é escreve o primeiro elemento, volta, verifica que não é e escreve o primeiro sem o primeiro, até verificar que é vazia, vai começar a ler o primeiro, que é o último elemento escrito, até verificar que é vazio


BALÃO
B
ALÃO
A
LÃO
L
ÃO
Ã
O
O
O
O
O
ÃO
Ã
LÃO
L
ALÃO
A
BALÃO
B

ativ8: quando encontra com uma lista primeiro verifica se é vazia, como não é, pega o último e lê o último sem o último e verifica se é vazia até que é e então ele volta



Foi extremamente interessante as respostas apresentadas por este participante.

Nas respostas dadas às atividades gráficas ele não havia feito uso de nenhum esquema de representação gráfica da execução. É importante frisar que foi o participante que mais tempo dispendeu junto ao sistema na apresentação feita, chegando inclusive a editar um procedimento recursivo que copiou de um livro. A princípio, para mim era o participante com maior dificuldade, e muita resistência, julgando-se incapaz de atingir qualquer resultado.

Pelas respostas que deu, verifica-se que foi percebido o processo de retorno, apesar de serem verificados erros nos valores dos parâmetros e respectivas operações de escrita. O mais interessante é a forma de representação utilizada que é bastante semelhante a representação da árvore de ativação do sistema. Isto de alguma forma mostra que a representação foi para ele bastante significativa, pois de posse dela o participante conseguiu se aproximar bastante do entendimento desejado. Novamente, o

objetivo de instrumentar o aluno de forma a auxiliá-lo a pensar parece ter sido alcançado.

Um aspecto deve ser observado quanto as representações utilizadas no sistema. Pode-se perceber que elas não são adequadas para serem utilizadas fora do sistema, e isto é realmente intencional. O objetivo não é que os alunos aprendam um outro padrão de representação e sim que de posse dos conceitos construam suas próprias formas de representação.

Depois de feitas as descrições, os alunos observaram a execução dos procedimentos junto ao sistema. No decorrer desta observação, foram propostas alterações nos procedimentos de forma a certificar o entendimento e estas sempre foram efetuadas com sucesso.

9.5 Introdução do Comando ENVIE

Muitas atividades foram desenvolvidas, seguindo uma metodologia semelhante à descrita no capítulo anterior, principalmente com relação à introdução do comando Envie.

Persisti introduzindo este comando através da escrita de predicados.

A mesma estratégia de utilizar primeiro o comando Escreva e depois substituí-lo pelo Envie foi adotada pelos alunos. Concluí que de certa forma isto é bastante positivo, pois consegui enfatizar de modo bastante significativo a diferença entre uma operação e um comando. O sistema foi bastante utilizado. Percebi que toda informação verbal que passava sobre o funcionamento de um procedimento, somente era assimilada quando a execução era seguidamente observada.

O nível de projetos desenvolvidos pode ser exemplicado pelos problemas a seguir:

Escrever um predicado que verifique se duas palavras são anagramas ou não:

```
ap anag :x :l
se não sãoiguais ( nel :x ) ( nel :l ) [ envie "falso ]
se não ( eelemento ( pal ( pri :x  )  "  ) :l ) [ envie "falso ]
envie anag sp :x tira :l pri :x
fim
```

```
ap tira :l :x
se :x = ( pri :l ) [ envie sp :l ]
envie  pal ( pri :l ) ( tira sp :l  :x )
fim

ap éelemento :x :l
se évazia :l [ envie "falso ]
se saoiguais :x pri :l [ envie "verd ]
envie eelemento :x sp :l
fim
```

Fazer um procedimento que coloque um número em uma lista de números ordenada de forma a não desordená-la:

```
ap colocar :x :l
se évazia :l [ envie ji :x :l ]
se  pri :l  =  ( :x ) [ envie :l ]
se ( pri :l ) > ( :x ) [ envie ji :x :l ]
envie colocar :x sp :l
fim
```

O uso do sistema também possibilitou que os alunos ganhassem uma estratégia de como testar um procedimento. Como eles a cada passo do desenvolvimento de um projeto recorriam ao sistema para verificar o funcionamento e o faziam para os mais diferentes valores com bastante facilidade, conseguiram depreender quais eram os valores significativos. Por exemplo, para verificar o funcionamento de um procedimento recursivo, iniciavam sempre a execução com um valor que ocasionava a parada de imediato. Em seguida, verificavam para um valor que provocava um nível de recursão. Depois iam em um crescente de níveis e sempre procuravam verificar como era a execução de todas as possibilidades, no caso de um procedimento com muitos casos condicionais. Queriam observar como era a execução de cada caminho. Isto foi bastante significativo, pois esta estratégia de como testar um programa é muito ensinada em cursos formais e neste caso foi provocada, ou facilitada, pelo ambiente de depuração utilizado.

Ao final do curso, efetuei uma terceira fase de diagnóstico que descrevo a seguir.

9.6 Terceira Fase de Diagnóstico

Esta terceira fase de diagnóstico tinha dois tipos de atividade: descrever o funcionamento e escrever procedimentos.

Vou descrever os problemas propostos e algumas respostas que exemplificam as respostas obtidas.

A) Utilizando qualquer representação diga o que faz:

```
ap um :p
se évazia :p [esc :p] [ pare]
esc :p
um su :p
fim
```

Foi geral a utilização de um exemplo, que mostrasse o funcionamento e todas as respostas foram corretas. Alguns exemplos de resposta são mostrados a seguir:

"O procedimento transforma uma lista de n elementos em listas com n-1 elementos até acabar a lista" (SUJ1)

ex:

```
[1 2 3]
[ 1 2 ]
[ 1 ]
vazio
pare
```

"Escreve a lista que eu entro completa e vai tirando o último elemento dela e mostrando" (SUJ4)

ex:

```
? um [1 2 3 4]
1 2 3 4
1 2 3
1 2
1
```

"Dada uma lista; testa se é vazia; se não for mostra a lista e retira o último elemento da lista; quando for vazia mostra a lista e pára".(SUJ3)

"Dada uma lista [a b c d]

primeiro verifica se évazia; como não é mostra a lista [a b c d] e chama novamente o programa tirando o último; verifica se évazia, como não é mostra sem o último [a b c]; volta, e chama novamente o programa um sem mais um último, verifica se évazia, como não é escreve sem o último [a b]; volta, e chama novamente o programa um sem mais um último, verifica se évazia, como não é escreve sem o último [a b]; volta, e chama novamente o programa um sem mais um último, verifica se évazia, como não é escreve sem o último [a]; volta, e chama novamente o programa um sem mais um último, verifica se évazia, e é porque tirou o último elemento, então pára, mostra [] e fica voltando no fim, só que não faz nada porque não tem comando"(SUJ6).

Observando a última resposta, o usuário está executando mentalmente o procedimento. Isto corresponde a hipótese de que ao predizer o comportamento de um programa realmente a pessoa

executa um modelo mental do processo de execução. E é possibilitar a aquisição deste modelo, um dos objetivos do sistema, que parece foi alcançado. A resposta é bem detalhada, mas está correta. O mesmo processo descritivo este participante irá utilizar em outras respostas.

B) Utilizando qualquer representação diga o que faz:

```
ap dois :p
se évazia :p [esc :p] [ pare]
dois su :p
esc :p
fim
```

Também neste exercício, houve 100% de acertos. Algumas respostas são dadas a seguir:

"Dada a lista [a b c d]

primeiro verifica se é vazia, como não é, volta e chama novamente o programa dois sem o último dois [a b c]; verifica se a lista é vazia, como não é ,volta e chama novamente o programa

dois sem o último dois [a b]; verifica se a lista é vazia, como não é ,volta e chama novamente o programa dois sem o último dois [a]; verifica se a lista é vazia, como não é ,volta e chama novamente o programa dois sem o último dois []; verifica se a lista é vazia, como é pára e mostra a lista vazia; vai terminando e mostra as listas sem o último [a], [a b], [a b c], [a b c d]"(SUJ6)

"Dada uma lista; testa se é vazia; se não for tira o último elemento da lista e repete os passos iniciais até a lista ficar vazia

quando for vazia mostra [] e pára

mostra todas as voltas até a lista entrada, com cada vez um elemento a mais na lista(SUJ3)

ex: dois [a b c]

évazia mostra []

[a]

[a b]

[a b c]

"Vai na lista tirando o último até ficar vazia. Ai vai voltando e escrevendo os elementos do vazio até a lista toda"(SUJ5).

C) Utilizando qualquer representação diga o que faz:

```
ap três :p
se não évazia :p [ três sp :p pare]
esc :p
fim
```

Novamente foi obtido 100% de acertos. Algumas respostas podem ser exemplificadas:

"Testa se uma lista não é vazia
enquanto for verdadeiro retira o primeiro elemento da lista
quando for falso começa a mostrar as voltas"(SUJ2)

```
três [a b c]
três [b c]
três [c]
três [] ---> falso
mostra []
       [c]
       [b c]
       [a b c]
```

"Mostra o último elemento, o último e o penúltimo, ..., vai juntando no início os anteriores"(SUJ4)

```
três [ 1 2 3 4]
4
3 4
2 3 4
1 2 3 4
```

D) Utilizando qualquer representação diga o que faz:

```
ap cinco :p
col (nel :p)/2 "a
envie ação1 (int :a) :p
fim

ap ação1 :n :p
se :n = 0 [ envie :p]
envie ação1 :n-1 sp :p
fim
```

Este é um exercício bastante diferente de todos os que tinham sido desenvolvidos durante o curso. O que ele faz é devolver uma

lista sem a primeira metade dos elementos. Dois participantes não responderam. O restante respondeu corretamente. Exemplificando as respostas tem-se:

(SUJ1) para a lista [1 2 3 4]

a = 2

ação1 2 [1 2 3 4]
:n :p

1 [1 2 3]
:n :p

0 [1 2]
:n :p

Separa a lista de entrada em duas listas e envia a primeira parte.

Considerei esta resposta correta, dentro de meus objetivos, apesar do usuário ter indicado a primeira metade como resultado.

"dada uma lista esse procedimento devolve a última metade da lista" (SUJ2)

"retira da lista a parte anterior de elementos ao elemento correspondente a parte inteira de (numero de elementos /2)"(SUJ4)

ex: se a lista tem quatro elementos, tira os dois primeiros.

"vai tirando elementos da lista até chegar a metade. Se a metade não dá inteira pega a parte inteira. se a lista tem sete elementos tira até o 3. (SUJ5)

As respostas dadas a este procedimento foram bastante significativas, por ser um exercício de natureza bastante diferente dos usualmente utilizados como exercício. O processo que conduziu as respostas foi também bastante elaborado, fazendo uso de muitos exemplos e representando bastante bem o fluxo de execução. Não foram utilizadas formas gráficas de representação, apenas a chamada e retorno dos procedimentos com os respectivos parâmetros, de forma análoga a que mostrei no primeiro exemplo de resposta. Realmente os alunos estavam instrumentados de modo a poder pensar sobre um procedimento diferente. Nenhum resquício de padrão poderia ser utilizado.

E) Utilizando qualquer representação diga o que faz:

```
ap seis :p :x
se évazia :p [ envie :p]
se pri :p = :x [envie seis sp :p]
envie ji pri :p seis sp :p
fim
```

Havia sido desenvolvido no decorrer do curso um exercício semelhante que pedia para que fossem retiradas todas as vogais de uma palavra. Este retira todas os elementos de uma lista :p iguais ao valor do parâmetro x. Apenas um participante não respondeu, os demais responderam corretamente. Algumas respostas podem ser exemplificadas:

" retira da lista o elemento pedido"(SUJ2)

" O exercício é para averiguar se um elemento pertence a uma lista . Quando pertencer o elemento é retirado da lista e o valor emitido é a lista (na mesma ordem) só que sem o elemento."(SUJ3)

F) Escreva um procedimento que:

1) receba uma palavra de entrada e substitua toda ocorrência da letra A pela letra O

Dos seis participantes, quatro apresentaram a resposta correta que pode ser descrita por:

```
(SUJ4) ap sete :l :o
       se évazia :l [envie :l]
       se pri :l = "a [ envie pal "o sete sp :l]
       envie pal pri :l sete sp :l :o
       fim
```

Um apresentou um solução bastante próxima da correta, que foi à seguir:

```
(SUJ1) ap palavra :nome :a
       se évazia :nome [envie :nome]
       se sãoiguais pri :nome :a [envie "a]
       envie palavra pri sp :nome :a
       fim
```

E um não apresentou solução.

2) receba uma palavra de entrada e conte todas as ocorrências da letra **A**

Para este procedimento, três pessoas apresentaram uma solução correta, que foi:

```
(SUJ6) ap contar :x :n
       se évazia :n [ envie 0]
       se pri :x = :n [ envie 1 + contar sp :x]
       envie 1 + contar sp :x
       fim
```

Duas pessoas apresentaram uma solução bem próxima da correta, que foi:

```
(SUJ5) ap contar :l
       se évazia :l [ envie 0]
       se pri :l = "a [ envie 1]
       envie 1 + contar sp :l
       fim
```

E uma pessoa não apresentou resposta.

Resumindo, pode-se dizer que apenas uma pessoa não conseguiu efetuar estas atividades de escrever um procedimento. Isto sem dúvida é mais difícil de ser feito, tendo em vista que não estava sendo utilizado o computador. O resultado, como reflexo da média de aproveitamento em cursos deste tipo, pode ser considerado excelente.

9.7 Conclusão

De modo geral os resultados foram bastante satisfatórios, pois mostrou que tendo como base estes diagnósticos, 90% dos participantes teve pleno aproveitamento. Esta média em muito supera o rendimento de qualquer outro curso que tenha sido dado. Credito muito do resultado à motivação e facilidades providas pelo sistema, pois em outros cursos do mesmo porte a abordagem com relação as atividades a serem desenvolvidas não era muito diferente e os índices de aproveitamento giravam em torno de 40%. O único fator de mudança foi a introdução do sistema de representações.

Um aspecto que tornou este uso do sistema diferenciado, foi o total desconhecimento dos participantes das operações de

manipulação simbólica. Isto de certa forma facilitou, pois o uso do sistema pode ser introduzido mais cedo neste aprendizado. E pode-se concluir que quanto mais cedo for possibilitada a visibilidade do funcionamento das construções, mais fácil se torna seu uso.

Mais uma vez reforça-se o aspecto que o ambiente gráfico não favorece a aquisição de conceitos de programação importantes no sentido de facilitar a passagem para o paradigma funcional de programação. E desta experiência posso reafirmar que a maior dificuldade na mudança de paradigma está ligada a não formalização destes conceitos e não a nova maneira de pensar sobre um problema.

Capítulo 10

Quanto os Especialistas de Computação Dominam o Conceito de Recursão: Um Estudo de Caso

10.1 Introdução

Outro exemplo de utilização do sistema bastante interessante foi feito por um aluno com formação em computação e que estava fazendo pós-graduação em computação.

A idéia de trabalhar com este aluno foi uma consequência de uma atividade, descrita no Capítulo 2, que fiz com alunos do curso de Ciência da Computação. Através desta atividade pude constatar que mesmo alunos de um curso específico de computação apresentavam sérias falhas no entendimento de um processo recursivo e estas falhas não estavam sendo percebidas, dentro dos moldes como eram efetuadas avaliações. Fiquei então com muito interesse em utilizar meu sistema junto a este tipo de usuário, porém com os alunos de computação que participaram da atividade não foi possível, dado que eles não conheciam Logo.

O aluno envolvido neste pequeno estudo de caso tinha se graduado em computação e estava cursando o primeiro ano de um curso de pós-graduação em computação, a nível de mestrado. Ele conhecia Logo bastante bem e portanto se mostrou bastante

adequado para que eu pudesse desenvolver a observação que tinha interesse.

Trabalhei com ele durante um curto período de tempo, cerca de 10 horas em dias diversos. O método de trabalho foi bastante informal, não estanto inserido em nenhum esquema de curso e foi basicamente especulativo.

O trabalho como um todo foi dividido em duas partes. A primeira, objetivou fazer um diagnóstico do nível de entendimento do aluno sobre recursão, em três contextos distintos: utilizando a linguagem Pascal, dentro do Logo gráfico e dentro do Logo listas. Nesta primeira parte foi utilizada a sequência de atividades já usada nos cursos anteriores e que tinham se mostrado adequadas a este diagnóstico.

A segunda parte foi de uso do sistema, onde deixei que o trabalho fluísse mais por conta do aluno, sem nenhum esquema "a priori" de atividades. Com isto pretendia observar como o sistema seria utilizado por uma pessoa com este perfil, bastante diferente do perfil de "usuário alvo" do meu sistema.

10.2 Primeira Fase : Diagnóstico

Centralizei toda atividade de diagnóstico em torno do entendimento de recursão, procurando explorar os contextos familiares ao sujeito.

A seguir vou descrever com detalhe as atividades em cada um destes contextos.

10.2.1 Pascal

O aluno tinha um bom conhecimento de programação Pascal e inclusive, estava desenvolvendo um compilador recursivo descendente para a linguagem Pascal.

De início solicitei que ele me escrevesse em Pascal, alguns algoritmos clássicos de percurso de árvores, nas mais diferentes ordens. São algoritmos recursivos, cuja solução não recursiva é bastante complexa. Ele apresentou as soluções clássicas sem maiores problemas e demonstrou bastante segurança.

Em seguida apresentei o problema da Propagação de Gripe, o mesmo já utilizado na atividade com alunos de computação. A atividade apresentava três formas de procedimento para atualizar um campo de uma lista ligada. O enunciado do problema para facilitar o acompanhamento é reapresentado a seguir:

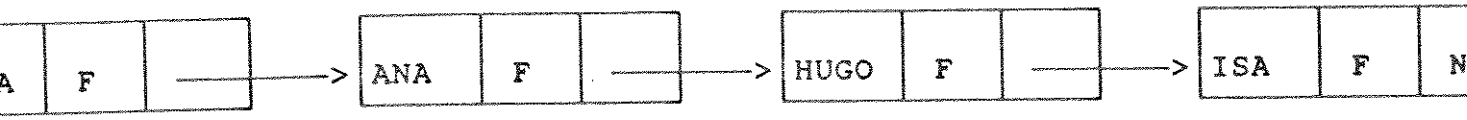
PRIMEIRA ATIVIDADE: RECURSÃO EM PASCAL

Deseja-se escrever um programa que faça a seguinte inferência:
"Se uma pessoa X tem GRIPE, então uma pessoa Y que beija X também tem GRIPE, e então a infecção se propaga para a pessoa que Y beija, e assim por diante"

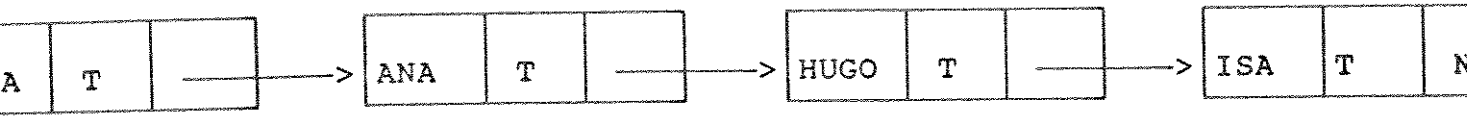
Dada a seguinte estrutura:

```
Type  p = ^reg;

      reg = record
            nome:string;
            gripe:boolean;
            beijo:p
            end;
```



Deseja-se fazer um procedimento que a altere para:



Foram apresentadas três soluções nomeadas de SOLU1, SOLU2 e SOLU3

Gostaria que voce considerasse cada solução por vez e respondesse:

a) se o procedimento faz ou não faz o desejado

b) se o procedimento faz, diga como ele faz (em suas próprias palavras) ou se ele não faz diga porque ele não faz (novamente com suas próprias palavras).

```
procedure SOLU1 (ap:p);
begin
  if ap^.beijo <> nil then ap^.beijo^.gripe := true
end;{SOLU1}
```

```
procedure SOLU2 ( ap:p);
begin
ap^.gripe:=true;
if ap^.beijo <> nil then SOLU2 (ap^.beijo)
end; {SOLU2}

procedure SOLU3 ( ap:p);
begin
if ap^.beijo <> nil then SOLU3 (ap^.beijo);
ap^.gripe:= true
end; {SOLU3}
```

RESPOSTAS DADAS PELO ALUNO:

a) SOLU1 definitivamente não faz. Só atribui true ao primeiro elemento da lista.

b) Como é para atribuir para toda a lista, este faz. Vai atribuindo até não ter mais para quem atribuir, ou seja, fim de lista (encontrar o NIL). A estrutura é recursiva.

c) Acho que não faz. Ele vai percorrendo a lista até o fim e então ele atribui só para último que tem campo NIL. Estou em dúvida, a não ser que aqui tenha alguma coisa de retorno de recursão e aí parece que funciona.

Surpreendentemente ele não reconheceu como correta a solução que utilizava o retorno da recursão para efetuar atualização na lista dando um padrão de resposta não muito diferente do que foi obtido com os outros usuários com os quais eu trabalhei. Observando somente esta resposta, pode-se dizer que ele possui o modelo de "loop" da recursão. Mas com este modelo ele não conseguiria resolver nenhum problema de percurso de árvores, onde são usadas operações no retorno da recursão. Isto é um paradoxo, e duas hipóteses puderam ser levantadas:

.o aluno se distraiu ao dar a resposta e portanto ela não tem nenhum significado

.o aluno realmente não entende completamente como funciona um processo recursivo e as soluções recursivas apresentadas por ele mais uma vez demonstram a aquisição de padrões de procedimentos recursivos.

Para mim, que estava junto ao aluno no momento em que ele deu as respostas a primeira hipótese parecia pouco provável, mas não podia ser descartada. As atividades seguintes vieram de certa forma confirmar que a segunda hipótese era a mais viável.

10.2.2 Logo Gráfico

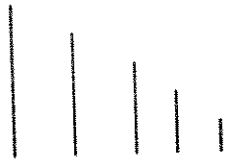
Em seguida iniciei as atividades com Logo. Comecei trabalhando com problemas gráficos. Aqui a proposta foi que ele desenvolvesse procedimentos gráficos, sem a exigência "a priori" de utilizar ou não recursão. Pretendia com isso observar até que ponto ele utilizava recursão e se usava ou não o retorno.

Para confirmar minhas idéias da atividade anterior, tinha como hipótese que o aluno utilizaria recursão de "rabo", mas não utilizaria operações no retorno.

Os problemas propostos e as respostas dadas pelo aluno são apresentadas a seguir:

SEGUNDA ATIVIDADE: RECURSÃO EM LOGO GRAFICO

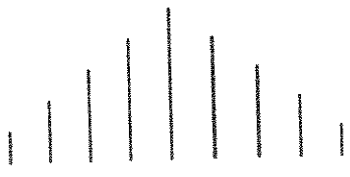
a) Fazer um procedimento que desenhe:



RESPOSTA DO ALUNO:

```
aprenda escada :i :f :inc :des
se i>=f  [ pf :i pf -:(i) pd 90
           un pf :des ul pe 90]
escada (:i - (:inc)) :f :inc :des
fim
```

b) Fazer um procedimento que desenhe:



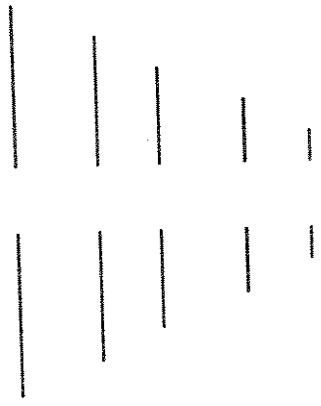
RESPOSTA DO ALUNO

```
ap volta :i :f :inc :des
escada :i :f :inc :des  {procedimento definido na atividade
fim                      anterior}


ap ida :i :f :inc :des
se :i <= :f [ pf :i pf -:(i) pd 90
              un pf :des pe 90 ul]
ida (:i + :inc) :f :inc :des
fim

ap escada2 :i :f :inc :des
ida :i :f :inc :des
pt -(:f)
volta :f :i :inc :des
fim
```

c) Fazer um procedimento que desenhe:



RESPOSTA DO ALUNO:

```
aprenda espelho  :i :f :inc :des :esp
se i>=f [ pf :i un pf :esp ul pf :i
          un pt 2* (:i) pt :esp pd 90
          pf :des pe 90 pf :inc ul]
espelho :i - (:inc) :f :inc :des :esp
fim
```

Fica clara a formação de computação do sujeito dada a preocupação que ele tem em todos os procedimentos, de parametrizar completamente a solução. Quando se trabalha com estes mesmos exemplos com pessoas que não são de formação

computacional, somente é utilizado como parâmetro o tamanho final ou inicial do traço. Isto é uma diferença, mas em termos de conceitos e estrutura, as soluções não diferem tanto quanto era esperado.

Efetivamente não foi utilizado o retorno da recursão. No procedimento B, o retorno deveria ser utilizado como consequência direta do procedimento A, caso o aluno conseguisse visualizar como este é efetuado.

No terceiro procedimento observe que o aluno denomina o programa de espelho, mas divide como unidade toda uma coluna, ou seja, a repetição é feita considerando uma coluna como base da repetição. Novamente, utilizando o retorno da recursão o efeito de "espelho" poderia ser descrito.

Através dos resultados obtidos nesta atividade, reforcei minha idéia de que realmente o aluno não possuia o conceito de recursão em seu sentido mais amplo, tratando recursão como uma repetição. Isto foi novamente reforçado através das respostas dadas na atividade de Logo listas.

É importante ressaltar que dado o tipo deste usuário ao qual foi proposta esta atividade ela foi eficaz. Com usuários que não sejam bons programadores acredito que este tipo de diagnóstico não vai ser eficaz, pois a obtenção de um resultado dentro do ambiente gráfico, não necessariamente iria envolver o uso dos conceitos pretendidos. Deve sempre ser considerada a característica empírica da programação no ambiente gráfico.

10.2.3 Logo Listas

As falhas conceituais foram também observadas quando foi efetuada a atividade de antever o funcionamento de procedimentos que efetuavam percurso em listas. As atividades e as respostas do aluno são listadas a seguir:

TERCEIRA ATIVIDADE - RECURSÃO EM LISTAS

A) Diga o que faz o procedimento:

```
ap um :k
se evazia :k [ pare ]
esc pri :k
um sp :k
fim
```

RESPOSTA DO ALUNO: escreve os elementos de uma palavra. Responde depois de testar com a palavra ANA. Faz o teste sem o uso de qualquer representação (tudo mental).

Como era esperado não apresentou maiores dificuldades em responder corretamente a este exercício.

B) Diga o que faz o procedimento:

```
ap dois :j
se evazia :j [ pare ]
esc pri :j
dois sp :j
esc ult :j
fim
```

RESPOSTA DO ALUNO: faz a mesma coisa que o anterior. Não, não. Tem uma diferença. Ele vai escrever o último elemento duas vezes. Por ex:

```
dois "bala          escreve b
                            a
                            l
                            a
                            a
```

O modelo de recursão como repetição fica claro nesta resposta, que não desconsidera o comando após a chamada recursiva, mas desconsidera os níveis de retorno.

C) Diga o que faz o procedimento:

```
ap três :l
se evazia :l [ pare ]
esc pri :l
três sp :l
esc pri :l
fim
```

RESPOSTA DO ALUNO: este escreve duas vezes o primeiro, depois duas vezes o segundo, e assim por diante:

Por ex: três [a b c] escreve a
a
b
b
c
c

Foi uma resposta bastante estranha, que acredito foi decorrente da resposta dada ao exercício anterior.

No exercício anterior a resposta foi que ele escrevia duas vezes o último. Mas aqui para escrever duas vezes o primeiro era preciso estar no primeiro, o que não ocorre no final da recursão. Então acredito que esta resposta foi a única que dava uma certa coerência à resposta dada ao exercício B. O mesmo vai acontecer com a resposta dada ao exercício seguinte:

D) Diga o que faz o procedimento:

```
aprenda quatro  :m
se évazia :m [ pare ]
esc ult :m
quatro su :m
esc pri :m
fim
```

RESPOSTA DO ALUNO: este faz o contrário do outro. Ele escreve o último e o primeiro sucessivamente até o fim da lista

por ex: **quatro [a b c]** escreve **c**

a

b

a

a

a

Portanto em todos os procedimentos foi mostrada muita confusão na idéia de como funcionava a recursão, principalmente com os procedimentos com volta de recursão.

Depois disso iniciei a fase de uso do sistema, que descrevo a seguir.

10.3 Segunda Fase: Uso do Sistema

A partir deste ponto ele passou a utilizar o sistema. Como era esperado não teve nenhuma dificuldade em utilizá-lo, pois já estava acostumado com ambientes semelhantes.

Mostrou-se bastante interessado e muito surpreso com a execução dos códigos dos procedimentos que havia trabalhado na última atividade. Depois de observá-los escreveu junto ao sistema alguns procedimentos recursivos muito utilizados no ensino convencional, como os listados a seguir:

Cálculo do fatorial, recursivamente:

```
aprenda teste
col pri line "n
esc fat :n
fim


aprenda fat :n
se :n < 1 [ envie 1 ]
envie :n * fat :n - 1
fim
```

Cálculo do fatorial não recursivo:

```
aprenda teste
col pri line "n
fat :n
fim

aprenda fat :n
col 1 "f
col 1 "aux
repita :n [ col :f * :aux "f col :aux + 1 "aux ]
esc :f
fim
```

Procedimento para juntar duas listas:

```
aprenda teste
col line "a
col line "b
esc append :a :b
fim
```

```
aprenda append :c :d
se évazia :c [ envie :d ]
envie ji pri :c append sp :c :d
fim
```

Procedimento para inverter uma lista:

```
aprenda teste
col line "l
esc inverso :l
fim


aprenda inverso :l
se évazia :l [ envie :l ]
envie jf pri :l inverso sp :l
fim
```

Todos estes procedimentos foram escritos com muita facilidade. Em todos eles ele efetuou a execução dentro do sistema no modo

passo-a-passo. Estava claro que ele sabia que os procedimentos funcionavam e estava interessado em ver como funcionavam.

Desta interação inicial com o sistema, rapidamente reformulou as respostas aos exercícios, e interessante que reformulou inclusive a resposta dada ao primeiro problema de Pascal, sobre a Propagação da Gripe.

O último procedimento padrão que ele executou foi o da Torre de Hanói, que é um problema recursivo clássico, que todos os alunos que passam por um curso formal de computação sabem escrever, mas que na realidade nunca entendem como funciona. Fica portanto claro o porque dele executar este procedimento.

Para verificar que realmente havia um entendimento do conceito propus alguns problemas que ele resolveu com bastante facilidade. A seguir, para exemplificar, vou relacionar alguns destes problemas.

A) Escreva um procedimento que verifique se um número é par, sem usar a operação de resto

RESPOSTA:

```
aprenda par :n
se :n = 2 [ esc "épar pare ]
se :n = 1 [ esc "éimpar pare ]
par :n - 2
fim
```

Em seguida perguntei: Se quizéssemos o predicado equivalente, qual alteração deveria ser feita?

RESPOSTA:

```
aprenda par :n
se :n = 2 [ esc "verd pare]
se :n = 1 [ esc "falso pare]
par :n-2
fim
```

Perguntei: Um predicado deve retornar como valor "falso e "verd e não escrever. Certo?

RESPOSTA:

```
aprenda par :n
se :n = 2 [ envie "verd ]
se :n = 1 [envie "falso ]
envie par :n-2
fim
```

Perguntei: Como deve ser usado este procedimento?

Resposta: Como um valor em outro comando. Por ex: ? esc par 10

B) Escreva um predicado que verifique se uma letra está ou não presente em uma palavra.

Primeira solução:

```
aprenda pertence :letra :pal
se pal = "  [pare]
se pri :pal = :letra [envie "verd]
envie pertence :letra sp :pal
fim
```

Perguntei: Quando vai dar o "falso ?

Resposta: quando rodar até o fim e não sair no ENVIE

Perguntei: E o que significa rodar até o fim?

Resposta: quando acaba o que comparar?

Segunda solução:

```
aprenda pertence :letra :pal
se pal = " [envie "falso]
se pri :pal = :letra [envie "verd]
envie pertence :letra sp :pal
fim
```

C) Escreva um procedimento que caso uma letra esteja presente em uma palavra conte o número de ocorrências

RESPOSTA:

```
aprenda freq :letra :pal
se :pal = " [envie 0]
se pri :pal = :letra [envie 1 + freq :letra sp :pal]
fim
```

Perguntei : Se ao invés desse procedimento eu quizesse contar o número de letras de uma palavra ?

RESPOSTA:

```
ap contar :pal
se :pal = " [envie 0]
envie 1 + contar sp :pal
fim
```

D) Escreva um procedimento que retire um elemento X de uma lista

RESPOSTA:

```
ap retirar :x :l
se évazia :l [ envie :l ]
se pri :l = :x [envie sp :x]
envie pal pri :l retirar :x sp :l
fim
```

E) Escreva um procedimento que retire todas as ocorrências da letra X em uma palavra.

RESPOSTA:

```
ap retirar :x :pal
se évazia :l [ envie "]
se pri :l = :x [ envie retirar :x sp :pal]
envie pal pri :pal retirar :x sp :pal
fim
```

Foi um processo de desenvolvimento bastante rápido quando comparado com os outros usuários que havia trabalhado. Ele já possuia um grande repertório de conhecimentos de programação e não estava portanto restrito as aplicações gráficas do Logo apesar de ter falhas conceituais semelhantes.

10.4 Conclusão

Alguns pontos merecem ser bem explorados a partir deste estudo de caso:

a) O aluno é sem dúvida um bom programador, tanto em Logo como em Pascal, mas não entende recursão completamente. Recursão é uma repetição condicional, sendo usada quando se quer repetir todo o procedimento a partir do início, com diferentes parâmetros. A recursão com volta é tratada com muita restrição e percebe-se que não é completamente entendida, sendo sempre evitada quando escreve procedimentos nunca vistos antes.

b) O trabalho com este aluno, fortemente confirmou minha hipótese de que o entendimento de recursão está profundamente ligado ao entendimento que as pessoas tem de como o processo é efetivamente executado pelo computador. Esta compreensão geralmente é errônea, pois não considera a recursão como cópias do procedimento que são ativadas e que devem ser desativadas (não existe a noção do processo recursivo). E para que este entendimento se efetive é realmente necessário que se dê a oportunidade de uma "visualização" do processo.

c) Tornar o processo explícito auxilia e o grau de auxílio é proporcional ao tipo de estrutura mental que o usuário dispõe. Esta estrutura mental é conseguida a partir da experiência do

usuário com programação. No caso dos alunos analisados nos dois capítulos anteriores, o modelo mental era extremamente falho e o sistema deveria permitir duas coisas: forte correção (quase construção) do modelo e aquisição do conceito. No caso deste aluno o modelo já é bom e o sistema facilita corrigir o modelo e consequentemente os conceitos.

d) Além disso, as falhas de entendimento não estão restritas aqueles usuários não especificamente qualificados em computação e geralmente advindos de outras áreas de formação. Portanto deve ser dada atenção a este fato, pois para os usuários com o perfil deste aluno é absolutamente indispensável a aquisição correta dos conceitos.

Capítulo 11

Discussão e Conclusões

Muitas conclusões já foram efetuadas e estão relatadas no decorrer deste trabalho. Acredito não ser necessário retomá-las e reescrevê-las. O objetivo deste capítulo é discutir uma série de pontos de caráter mais geral sobre a atividade de programação, fazendo sempre que necessário referência aos aspectos desenvolvidos e resultados obtidos no trabalho, que justificam o levantamento e discussão das idéias.

11.1 Introdução

Psicólogos da linha da Gestalt [Wertheimer,59] distinguem dois modos de aprender a resolver problemas: aprendizado conduzido ("rote learning") e o do entendimento. Por exemplo, com respeito ao aprendizado de Matemática, é feita frequentemente, ou pelo menos deveria ser feita, uma distinção entre "conseguir a resposta correta" e "entender o que se está fazendo".

Em um exemplo clássico, Wertheimer sugere que existem dois caminhos básicos para ensinar a uma criança como encontrar a área de um paralelogramo.

O primeiro método envolve traçar uma linha perpendicular, medir a altura da perpendicular, medir o comprimento da base e calcular a área usando a fórmula área = base * altura. É a isto que Wertheimer denomina de aprendizagem conduzida ou ainda, de modo mais contundente, de método "sem sentido", pois o estudante é levado a memorizar uma fórmula e um procedimento para utilizá-la.

O segundo método pede para o estudante explorar o paralelogramo visualmente até que ele perceba que é possível cortar um triângulo de uma das pontas, colocá-lo na outra ponta e formar um retângulo. Desde que os estudantes já saibam como calcular a área de um retângulo, o problema está resolvido. Wertheimer chama este método de "entendimento estrutural" ou "apreensão de relações significativas", pois o estudante ganha uma visão da estrutura de paralelogramos.

De acordo com Wertheimer, se for dado um teste envolvendo paralelogramos do mesmo tipo dos que são usados durante o processo de instrução, ambos os grupos de crianças irão obter bons resultados. Entretanto se for dado um teste de transferência, que envolva paralelogramos não usuais, os

"aprendizes conduzidos" irão dizer que não lhes foi ensinado aquele conteúdo, enquanto os "entendores" serão capazes de derivar a resposta.

Este exemplo sugere que quando se tem por objetivo o uso criativo de novas informações técnicas, é importante usar métodos e ambientes que favoreçam o entendimento. O que se pode observar é que o ensino de programação está conduzindo o aprendiz a adquirir uma série de padrões sem contudo se preocupar com a aquisição dos conhecimentos envolvidos nestes padrões. Acredito que não seja isto que queremos que seja feito com programação.

Cursos sobre programação são oferecidos em diferentes níveis: da escola primária ao nível profissional. Dado este contexto muitas questões aparecem:

. Como ensinar programação?

. O que é necessário ser ensinado? Em qual nível?

As respostas à estas questões não estão automaticamente disponíveis; é preciso conhecer mais sobre como estudantes em

diferentes níveis aprendem a programar, quais dificuldades eles encontram e como a instrução pode ajudá-los a vencer estas dificuldades.

Mas, é importante ter bastante claro que endereçar o problema de quão bem os estudantes estão aprendendo a programar é um assunto que merece muita atenção.

11.2 Aprender a programar é difícil

Será feita uma discussão para ressaltar a complexidade e as dificuldades que envolvem o aprendizado de programação, e consequentemente os aspectos que devem ser tratados quando do ensino de programação. O objetivo é mostrar que este domínio de conhecimento merece muita atenção tanto em métodos de ensino como em materiais instrucionais.

Programação é um complexo domínio de conhecimento e prática. O aprendizado de programação significa a aquisição de conceitos específicos de programação, adicionalmente às habilidades de solução de problemas mediada por uma ferramenta tecnológica.

A grosso modo, talvez pudéssemos categorizar as dificuldades do aprendizado de programação em algumas áreas básicas, todas elas com um alto grau de sobreposição:

.**Problema geral de orientação**: entender o porquê de programar, que tipos de problemas podem ser tratados e quais as vantagens em dispender tempo programando.

.**Entendimento da máquina notacional** : dificuldades em entender as propriedades gerais da máquina que se está aprendendo a controlar e relacionar o comportamento físico da máquina com a máquina notacional definida pela linguagem em que se está programando. Este é o foco principal do sistema que implementei.

.**Entendimento da notação da linguagem de programação**: dificuldades em dominar tanto a sintaxe como a semântica das construções. A semântica pode ser entendida como uma elaboração das propriedades e comportamento da máquina notacional, mencionada no ítem anterior. Como o entendimento da semântica está muito relacionado com o entendimento da máquina, que é difícil de ser obtido dentro dos sistemas usuais de programação,

o que ocorre é que os alunos focalizam o aprendizado de programação nos problemas de sintaxe.

.Dominar as características pragmáticas de programação: saber como especificar, desenvolver, testar e depurar um programa. Adquirir estratégias de como isto pode ser feito utilizando as ferramentas que se tem disponível. Neste aspecto é que são centrados tanto o ensino em sala de aula como os tutoriais que ensinam programação.

A específica caracterização de programação como uma situação de solução de problemas pode ser resumida em alguns pontos:

. o objetivo da atividade não é somente produzir a solução, como em situações clássicas de solução de problemas, mas também tornar explícito o procedimento que produziu a solução

. a atividade é altamente intermediada por uma ferramenta tecnológica para a qual o sujeito tem que construir uma representação funcional

. a elaboração e expressão de um procedimento dentro de um "sistema de código de ação" necessita da aquisição e uso de conceitos de programação específicos como variáveis, parâmetros, procedimentos e subprocedimentos, recursão, etc.

Para exemplificar o nível de dificuldade no entendimento desses conceitos computacionais de programação, vou discutir a problemática relativa a dois conceitos: variável e recursão, sendo que recursão naturalmente envolve os conceitos de fluxo de controle e procedimentos.

. **Variável**

No estudo de modelos mentais existem pesquisas que revelam que as pessoas usam analogias para ajudá-las a estruturar domínios não familiares [Gentner, 1983a] . Por exemplo, pessoas discutem sobre corrente elétrica usando como modelo o tráfico de automóveis ou em termos de fluxo de água. Isto leva à definição de modelos conceituais de analogia [Norman, 1983], que como já discuti em capítulos anteriores, são fracos para o caso de conceitos computacionais.

Muitos dos erros de principiantes com variáveis são causados pela transferência do conceito de variáveis do domínio algébrico para o dominio computacional.

Dentro do paradigma funcional de programação esta transferência não causa conflitos, pois somente problemas muito sofisticados exigem o uso de variáveis como são conceituadas em programação. A problemática está concentrada no paradigma procedural de programação, onde variável tem que ser tratada como um novo conceito.

Variável do ponto de vista de programação, é definida como um endereço de memória ao qual se atribui um nome e um valor. Mas esta formulação não é suficiente para analisar o conceito em seu significado funcional. Na realidade quando o valor de uma variável muda, sua nomeação e seu relacionamento funcional com os outros elementos do programa são invariantes.

Para o estudante, pode-se distinguir quatro formas de atribuição, quando se opera com variáveis:

1. atribuição de um valor constante

```
atr "a 3
atr "x "banana
```

2. atribuição de um valor calculado

```
atr "a  5 * 7
```

3. duplicação

```
atr "b :a
```

4. acumulação

```
atr "x  :x + 5
atr expox   :expox * :x
```

O conceito de variável é total em programação no último caso, nomeado acumulação. Nos outros casos o estudante pode simplesmente utilizar o conceito familiar de variável e igualdade, ou seja, um valor assinalado a uma variável. Nesta mesma amplitude pode-se inserir o uso de variável como parâmetro, que como já afirmei em capítulos precedentes não dá o conceito mais amplo de variável.

O entendimento e a construção de representações simbólicas associadas com a forma de acumulação leva a ser necessário considerar a variável em sua definição temporal: " o valor da variável :expox no passo n+1 é igual ao valor da variável :expox no passo n multiplicado ao valor da variável x". O estudante tem que designar com o mesmo nome tanto o valor precedente como o atual . Portanto o modelo matemático de variável e da relação de igualdade constituem para o iniciante um modelo inicial mas não suficiente para operar com variáveis em programação.

. Recursão

Notacionalmente, um procedimento é recursivo quando chama a si mesmo.

Para entender como procedimentos recursivos trabalham em Logo, e em qualquer outra linguagem, é preciso entender :

1. A regra de execução sequencial, linha a linha, de um programa. Quando um procedimento é chamado o controle é passado adiante para o procedimento chamado e em seguida, é devolvido para o procedimento chamante, no ponto da chamada.

2. Quando um procedimento é executado ele é executado linha a linha, até seu final, ou seja até encontrar o comando Fim. O comando Fim portanto significa:

A) término da execução de uma unidade lógica do programa

B) retorno ao procedimento chamante desta unidade.

3. Existem algumas exceções desta execução linha a linha. Duas importantes, no caso do Logo, são o comando Pare e Envie, que ocasionam o retorno ao procedimento chamante. Funcionalmente significam um salto ao comando Fim mais próximo e no caso do comando Envie, também o retorno de um valor ao procedimento chamante.

É uma prática pedagógica sensível basear o entendimento de procedimentos recursivos no entendimento de procedimentos iterativos.

Anzai e Uesato [Anzai,1982] concluiram que escrever primeiro funções iterativas facilita escrever funções recursivas, e o inverso não vale. Eles afirmam ser difícil interpretar chamadas

recursivas dentro de procedimentos e aprender iteração primeiro ajuda com recursão, porque iteração provê um primeiro modelo de como é feita uma chamada recursiva. O perigo está em não evoluir deste modelo inicial para poder tratar recursões que não sejam "de rabo".

Em Logo e em linguagens similares, esta prática pedagógica não é muito viável, dada a não existência de comandos repetitivos condicionais. É por demais artificial forçar a escrita de procedimentos não recursivos quando a repetição é condicional. Deve-se utilizar inúmeros artifícios como, por exemplo, o uso de parâmetros fantasmas que controlem a repetição.

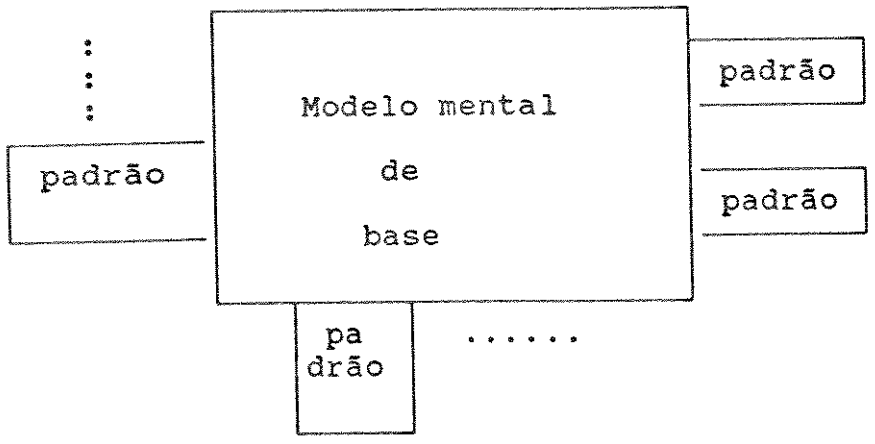
Mas mesmo em linguagens, como o Pascal, onde é viável se iniciar recursão a partir de analogia com o processo iterativo condicional, esta prática tem levado à uma compreensão errônea sobre recursão, como pode ser extensamente observado nas análises efetuadas no decorrer deste trabalho. Isto ocorre porque iteração não envolve o conceito de procedimento e consequentemente o modelo de cópias necessário ao entendimento de chamadas recursivas não é provido.

11.3 Modelo mental sobre computador e conceitos computacionais é falho

O objetivo desta parte da discussão é mostrar o tipo de modelo mental que os alunos vem desenvolvendo de programação. Além disso ressaltar que os métodos de ensino tradicionais não estão conseguindo aperfeiçoar este modelo e por consequência alternativas devem ser buscadas. Vou me ater aos aspectos de entendimento de recursão, que foi o aspecto em que centrei a fase experimental de meu trabalho. Mas acredito que o que foi observado pode ser extendido para programação de um modo geral.

Os protocolos que levantei revelaram que sujeitos possuem um modelo mental pobre de recursão e somente com um modelo adequado da funcionalidade se aprende a programar.

Deste trabalho pude concluir que podemos representar o modelo mental que os estudantes possuem de programação através do seguinte esquema:



Com os materiais instrucionais utilizados e o "feedback" que é obtido pelo ambiente de programação, os estudantes são capazes de desenvolver um modelo mental de base limitado e falho em muitos aspectos. Todos os programas cujo comportamento não consegue ser explicado através deste modelo de base são "agregados" como padrões.

Como o modelo de base não sofre alteração (evolução), somente são desenvolvidos e entendidos os procedimentos que possam ser explicados e os que "encaixam" em algum padrão. O modelo de base para recursão, quando existia, incluia recursão como repetição, e os alunos possuiam padrões de procedimentos recursivos, com

finalidades específicas, que envolviam retorno. Caso fosse solicitado um procedimento, que envolvia retorno de recursão e que fosse de natureza diversa da dos padrões agregados, não havia entendimento. Para exemplificar, pode-se verificar o protocolo descrito no Capítulo 10, onde o aluno de computação naturalmente resolvia sofisticados problemas de percuro em árvores, mas não conseguia identificar como correto um procedimento bastante simples de atualização de campos de uma lista, quando esta atualização era feita no retorno da recursão.

Estes aspectos sugerem a discussão sobre o porque dos recursos instrucionais utilizados tradicionalmente não estarem facilitando a aquisição de um modelo mental adequado de programação.

11.4 Recursos de Ensino não são Adequados

Como recursos de ensino vou considerar os métodos utilizados em sala de aula com giz e quadro-negro, os ambientes de programação e os tutoriais.

Quando se analisa as propostas de ensino apresentadas em

livros texto de cursos de programação, claramente se observa a tendência em fazer com que os alunos desenvolvam padrões. São enfatizadas estruturas de programa específicas para determinadas tarefas, como somadores, contadores, etc. Exite uma preocupação clara em nomear estes trechos e referí-los quando novos problemas os envolvem.

Por exemplo, Atwood e Ramsey [Atwood, 1978] afirmam que programadores experientes descrevem um segmento de código como o a seguir:

```
sum := 0;
for i:=1 to n do
 sum=sum+x(i);
```

Como "cálculo da soma de um array x". Um programador experiente tem um "schema" para esta tarefa e é capaz de gerar uma variedade de linhas de código para alcançá-lo. E o objetivo do ensino de programação é conduzir o aluno a este "status" de perito, através da aquisição destes esquemas.

Inclusive, Atwood e Ramsey [Atwood, 1980] afirmam que "é possível ensinar explicitamente os "chunks" ou "schemata" superiores envolvidos na programação de computadores. A nomeação explícita de "schemas" básicos deveria fazer parte de currículos de programação". Não é ressaltada a importância de analisar qual o conteúdo conceitual (processo iterativo, variável, acumulação, etc.) envolvido nestes "schemas", parecendo simplesmente uma atividade de nomear para depois reconhecer, da mesma forma que se nomeia objetos de um modo geral.

Não é considerado pelos defensores do ensino de padrões o aspecto central do uso adequado destes esquemas. O potencial uso destes planos implica em que a pessoa efetue uma transferência de conceitos e habilidades através de domínios, e isto depende da capacidade de deteção de similaridades entre o conhecido e o novo domínio. Isto só pode ser conseguido se for enfatizado o conteúdo conceitual envolvido nos esquemas.

Existem, é claro, tentativas de enfatizar ou auxiliar a obtenção destes conceitos básicos envolvidos em qualquer programa. Algumas foram analisadas no Capítulo 2, como por

exemplo as representações que são utilizadas para tentar espelhar o processo de execução. Estas tentativas não estão dando os resultados desejados, por não mudarem a essência do que é mostrado. O que é feito é geralmente um pouco mais do mesmo ou o mesmo com uma nova roupagem. Isto é semelhante as tentativas de ensinar fração usando formas fracionárias, como uma pizza por exemplo.

Os ambientes de programação, juntamente com as linguagens, também não dão o "feedback" necessário para que o aluno entenda o processo. As tentativas de dar algum auxílio, geralmente se restringem a uma extensa relação de valores de variáveis, que para o aluno entender é preciso que ele já conheça o processo. Sabe-se que visibilidade é um aspecto importante no aprendizado. Os principiantes precisam ver partes selecionadas do processo em ação. Quando esta visibilidade existe, observa-se progresso, como é o caso do Logo geométrico.

O uso do computador como auxilio na forma de tutoriais, é novamente um pouco mais do mesmo. É o ensino tradicional, feito pelo computador e portanto mais empobrecido. O uso de tutoriais

acredito que conduz mais fortemente a criação deste modelo de padrões agregados, pois o universo de problemas tratados é ainda menor e portanto a possibilidade de abstrair conceitos é bem menor.

O que está sendo perdido neste processo de ensinar através de uma enorme quantidade de exemplos, é focalizar à atenção nos conceitos básicos e que não são muitos: variável, iteração, procedimento/operação, fluxo de execução e recursão. Além disso, ao utilizar o computador como auxílio, não se pode perder que a aquisição de conceitos envolve um ciclo de fazer, refletir sobre o que fez, depurar, fazer, refletir,.... O computador deve ser inserido neste processo de aprendizado favorecendo, e não ensinando, que as pessoas possam individualmente através da exploração formar seu próprio modelo do conceito. Afinal, nunca nos foi ensinado sobre gatos e sobre o que os diferencia de cachorros. É preciso deixar o objeto programação passível de ser manipulado, de modo que o aprendiz possa interagir com esses objetos computacionais do mesmo modo que interage com animais e é capaz de aprender a diferença entre gato e cachorro.

É preciso dar visibilidade ao processo de execução e isto deve ser feito por quem efetivamente executa o programa, ou seja, o computador. E a visibilidade deve ser dada ao programa que o aluno fez. O processo de fornecer representações diferentes das usuais deve levar em conta que o computador possui características que o qualificam como a melhor mídia para provocar estas mudanças. As representações devem ser dinâmicas, como o processo que estão tentando espelhar. O ambiente deve ser interativo. Não interessa fornecer a representação de um procedimento padrão e não permitir a interação do usuário, na definição e visualização de seus próprios procedimentos. Isto é que torna diferente e válido o uso do computador como ferramenta auxiliar ao ensino, destacando-o sensivelmente de outras mídias como o vídeo na produção de um filme, ou de um livro em sua forma estática. E foram estas características que basearam o sistema que implementei.

11.5 O que foi feito e como isto ajudou à compreensão dos modelos que as pessoas tem de conceitos computacionais

O que objetivei no desenvolvimento do sistema e na forma como conduzi o uso, foi que através de uma interação mais efetiva com

um ambiente que explicitasse o processo de execução de programas, fossem adquiridas invariantes conceituais de forma a possibilitar a construção de um modelo mental adequado ao aprendizado de programação.

Esta abordagem de aprendizado que utilizei na concepção do sistema está de acordo com a teoria de desenvolvimento cognitivo de Piaget [Piaget, 1970]. Segundo esta teoria, entender um objeto é um processo dinâmico de redescrever e reinventar o objeto. E este processo se efetiva através da atuação do sujeito sobre o objeto. Se o objeto é programação, o aprendiz deve trabalhar com o programa "reinventando-o". Cabe ao instrutor tornar o processo explícito e o objeto mais palpável para ser entendido.

Utilizei o computador por achar que as formas de representação utilizadas tradicionalmente como os diagramas de execução, brincar de computador, árvores de ativação, etc., precisavam ser mudadas na sua essência. Do estático deveria-se passar para o dinâmico, pois o processo de execução de um programa é dinâmico. Do mostrado em um papel ou feito por um professor, deveria ser feito pelo próprio sistema que executa o processo, o computador.

E a representação ou visibilidade deveria ser dada ao programa feito pelo aluno e não para um exemplo padrão feito por um perito. E o usuário deveria ter meios de interferir no processo, controlando a forma como gostaria de observar e para quais exemplos.

Os resultados do uso confirmaram minha hipótese de que a visibilidade dada através de sistemas dinâmicos, realmente instrumenta as pessoas a pensarem sobre programação. Algumas conclusões devem ser retomadas:

. Entender como funcionam as construções de uma linguagem não é suficiente para entender programação. A maioria das pessoas observadas não conseguia a princípio sintetizar o funcionamento de um procedimento. Sempre que eram solicitadas a dizer o que um procedimento fazia, efetuavam uma leitura comando a comando, explicando-os isoladamente. Para entender programação é preciso entender que quando agrupados, na forma de um programa ou de um procedimento, os comandos deixam de ter um significado e estrutura descontextualizada. Este aspecto deve ser enfatizado no ensino de programação, pois sem isto nunca o aluno irá conseguir

adquirir estratégias de solução de problemas como a de dividir um problema em partes. Portanto, não adianta efetuar um extensivo trabalho na área de solução de problemas se o entendimento do processo de execução está falho. O ambiente de programação pode auxiliar neste aspecto tornando o processo mais explícito, nos moldes do sistema que desenvolvi.

. Programar sob o paradigma procedural é a melhor maneira de se iniciar programação. Acredito que isto ocorre por duas razões. Primeiro, por ser este o paradigma mais próximo do real funcionamento do computador, e por consequência leva o aluno a criar um modelo mental mais adequado de como o computador funciona. Segundo, por ser este o paradigma mais próximo de como as pessoas atuam no dia-a-dia. É muito eficaz a metáfora de ensinar uma nova palavra ou um novo comando ao computador, explicando o seu significado, e que uma vez ensinado pode ser utilizado sem precisar redefinir. Tendo este aspecto deve-se levar em conta que mudar um paradigma de programação envolve muito mais que mudar a estratégia de resolver um problema. Irá envolver a mudança do modelo mental que a pessoa faz do comportamento do computador. Isto deverá ser reformulado para que

a pessoa possa controlar a máquina efetivamente sob o novo paradigma. Portanto, toda mudança de paradigma também deverá envolver a invenção de um ambiente de representações que favoreça este novo entendimento da máquina notacional que está sendo desenvolvida.

. O entendimento de recursão, fluxo de execução, variáveis globais e locais, pilha de execução, etc. está profundamente ligado ao entendimento que as pessoas têm de como o processo é efetivamente executado pelo computador. E para que este entendimento ocorra é absolutamente necessário que se dê oportunidade de visualização do processo. Um exemplo claro é o trabalho efetuado com o aluno de computação, descrito no Capítulo 10. Este aluno é sem dúvida um bom programador, mas não tinha elaborado completamente o conceito de recursão, pois a recursão com volta nunca era utilizada. Escrevia muitos procedimentos padrões que envolviam volta, mas não conseguia explicar seu funcionamento, e quando a proposta era de um procedimento novo nunca visto, ele resolvia o problema mas nunca utilizava o retorno da recursão. O processo com que ele utilizou o sistema que implementei não foi para auxiliar no desenvolvimento de novos

programas, mas sim para visualizar o processo de execução de programas que ele já conhecia. E com isto ele reformulou muitas das soluções já apresentadas, achando que utilizar a recursão com volta deixava o "programa mais claro". Portanto tornar o processo explícito auxilia, e evidentemente o grau de auxílio é proporcional ao tipo de estrutura mental que o usuário dispõe em decorrência de sua experiência prévia com programação. Com os outros usuários também foi comprovado este auxílio, mas em menor grau. Com estes usuários, seria necessário um trabalho mais prolongado, onde mais procedimentos pudessem ser desenvolvidos e observados, para mais claramente abstrair as características invariantes do processo de execução, que deverão compor o modelo mental.

Portanto é preciso pensar em alternativas ao usual. Novas ferramentas devem ser adicionadas ao ensino, levando em conta os processos cognitivos envolvidos no aprendizado de programação. Quanto ao nível de entendimento da máquina notacional, acredito que o esquema de representações computacionais dinâmicas que espelhem o funcionamento da máquina através de um modelo conceitual coerente com o modelo mental que queremos seja

construído pelo aluno, é a melhor alternativa. Claramente antevejo um maior aprofundamento na pesquisa nesta área e é o que discuto na conclusão deste trabalho.

11.6 Conclusão

O que pretendo é fazer uma reflexão sobre tópicos de pesquisa que foram abertos a partir do estudo e das conclusões que realizei no decorrer deste trabalho.

. Quanto a modelos

De modo geral, mais pesquisa precisa ser feita para determinar o efeito específico de modelos no que está sendo aprendido, e para determinar as características de um modelo conceitual adequado. Modelos conceituais são ferramentas para o entendimento ou ensino de sistemas. São inventados para prover uma representação adequada do sistema em uso ou que está sendo ensinado, consistindo a base do modelo mental que se deseja que o usuário forme do sistema. Idealmente, quando um sistema é construído, seu projeto deveria ser baseado no modelo conceitual.

O modelo conceitual deveria governar toda a interface humana com o sistema e dessa maneira a imagem do sistema que é vista pelo usuário seria consistente, coesa e inteligível. O que geralmente se tem é uma imagem do sistema distante do modelo conceitual. Portanto, se instrutores e materiais instrucionais do sistema ensinam ao usuário o modelo conceitual de base, e se a imagem do sistema é consistente com este modelo, o modelo mental do usuário será consistente. Como designers, é nossa obrigação desenvolver sistemas e material instrucional que ajude o usuário a desenvolver um modelo mental mais coerente e mais útil. Como professores é nossa obrigação desenvolver modelos conceituais que possam ajudar o aprendiz a desenvolver modelos mentais adequados e apropriados. Determinar um modelo útil, naturalmente envolve o entendimento do tipo de conhecimento que é vivenciado pelo principiante, quais as dificuldades fundamentais e de que forma podemos alterar tanto a metodologia como os instrumentos utilizadas no ensino de programação.

. Aprofundar o estudo no domínio de programação

Os resultados que obtive demonstraram que existem falhas conceituais sérias no aprendizado de programação e que são

difíceis de serem superadas com metodologias tradicionais. Também demonstrei que um sistema que torne mais visível o processo de execução de um procedimento pode ser útil na aquisição destes conceitos. Em suma, mostrei que um sistema com as características do que implementei é usável para atender os objetivos de melhorar o entendimento de conceitos computacionais. Muitos pontos estão ainda em aberto, considerando-se o domínio de programação.

Como afirmei, demonstrei que o sistema é usável, se ele realmente favorece o aprendizado é uma questão em aberto. Um experimento, talvez com grupos de controle e bem esquematizado do ponto de vista metodológico segundo os padrões da psicologia experimental pode prover uma resposta a esta questão.

A natureza do conhecimento e o processo de criação deste modelo mental que irá guiar a interação do usuário com o computador deve ser efetivamente analisada. Tendo-se bem definido o papel de modelos mentais o efeito deste estudo pode ser expandido, por exemplo, para a construção de tutoriais mais efetivos, para a definição do que representa interação amigável, para a definição de quais características deve ter uma linguagem para principiantes, etc.

Entendendo melhor a natureza destes modelos mentais, o sistema deve ser avaliado no sentido de verificar se realmente é eficaz no sentido de auxiliar a construção de um modelo mental adequado. Se ocorre alguma alteração no modelo mental qual é esta alteração? Como ela ocorre? Caso exista um modelo mental adequado adquirido, ele é tranferível para outros contextos?

Outro ponto que deverá ser avaliado no sistema diz respeito as características de design. Deve-se avaliar se as características presentes no sistema são suficientes ao entendimento desejado. Isto deverá ser embasado no estudo mais aprofundado de modelos mentais, mencionado anteriormente. Com isso novas ferramentas ou representações podem ser adicionadas ao sistema, ou alguma das existentes pode ser suprimida. Algumas extensões ao sistema já estão sendo efetuadas, como decorrência de algumas observações que efetuei do uso. Toda parte de deteção e recuperação de erros está em desenvolvimento, visando com isto ganhar independência do sistema MSX, e proporcionar o desenvolvimento "visual" totalmente junto ao sistema. Com isso acredito que o sistema irá se tornar uma importante ferramenta de depuração, podendo fornecer esquemas de ambientes ideais de depuração. Outra característica que está

sendo desenvolvida é no sentido de prover representações de mais baixo nível, ou seja, dar visibilidade ao nível de um comando, mas diferente do que é feito no modo interativo ou direto de muitas linguagens, pois o funcionamento do comando será contextualizado. Esta última alteração decorreu diretamente da observação do uso do sistema, na fase experimental de meu trabalho. Acredito que um maior entendimento de que procedimentos que descrevem operações, funcionalmente são valores ou objetos da linguagem, será ainda mais favorecido quando for inserido no sistema um módulo de representações de mais baixo nível, que mostrem como é executado, passo-a-passo, um comando ou operação.

. Ampliar para outros domínios de conhecimento

Finalmente, o uso de sistemas computacionais dinâmicos de representações deve ser avaliado em outros domínios de conhecimento que também envolvam conceitos decorrentes do entendimento de processos abstratos. Estudar novas formas de representação que possam ser utilizadas em áreas de Ciência onde conceitos como equilíbrio, pressão, temperatura, etc., são

extremamente dependentes do processo onde ocorrem. Uma área que me interessa pessoalmente e que inclusive originou meu interesse por este tema de pesquisa, é Eletricidade. A idéia é procurar outras formas de representação (cores, som, analogias com outros domínios, etc.), diferentes, ou englobando, da tradicional, que auxiliem as pessoas a entenderem conceitos como voltagem, corrente alternada, corrente contínua, potência, etc.

E em todas estas áreas, deve ser avaliada a importância de modelos, como são criados, qual sua natureza, para se poder construir representações diferentes das usuais, sistemas cognitivamente eficazes e portanto modelos conceituais dinâmicos e úteis.

Bibliografia

ABELSON, H. e diSESSA, A. (1981)"Turtle Geometry: The Computer as a Medium for Exploring Math", MIT Press, Cambridge, MA

ACKERMANN, E. (1991) "The Clinical Method as a Tool for Rethinking Learning and Teaching", Paper presented at the Annual Meeting of the American Research Association, Chigago

ADAM, A. et al (1980) "A System to Debug Student Programs", Artificial Intelligence, 15, pp. 75-122

ANDERSON, John R. (ed.) (1981) "Cognitive Skils and Their Acquisition", Lawrence Erlbaum Associates, Publishers

ANDERSON, J. R. e SAUERS, R. (1984)"Learning to Programming Lisp", Cognitive Science, 8, pp. 87-129

ANDERSON, J. R. e RIESER, B. J. (1985) "The Lisp Tutor" BYTE, vol. 10, n. 4

ANDERSON, John R. e SKWARECKI, Edward (1986) "The Automated Tutoring of Introductory Computer Programming", Communications of the ACM, vol.29, n.9, pp.842-849

ANDERSON, J. R., BOYLE, C. et alli (1990) "Cognitive Modeling and Intelligent Tutoring", Artificial Intelligence, vol.42, n.1, pp.7-49

ANZAI, Y. e UESATO, Y. (1982) "Learning Recursive Procedures by Middleschool Children", Proceedings of the Fourth Annual Conference of the Cognitive Science, pp.100-102

ATWOOD, M. E. e RAMSEY, H. R. (1978) "Cognitive Structure in the Comprehension and Memory of Computer Programs: An Investigation of Computer Programming Debugging" ARI Tech. Rep. TR-78-A210, Science Applications, Englewwod, Colo., August

BARANAUSKAS, M. C. C. e SILVA, H. V. R. C. (1986) "O Computador: Um Novo Super-Herói", Cartgraf Editora Ltda, SP, Brasil

BARANAUSKAS, M. C. C. (1991) "Procedimento, Função, Objeto ou Lógica?", Anais do 8th ICTE Joint Conference, Ontario, Canada

BARR, A. et al (1976) "The Computer as Tutorial Laboratory: The Stanford BIP Project", International Journal of Man-Machine Studies, 8, pp. 567-585

BARSTOW, David R. et al (eds) (1984) "Interactive Programming Environments", McGraw Hill Book Company, New York, EUA

BODEN, Margaret A. (1985) "Jean Piaget", edited by Frank Kermode, Penguin Books

BONAR, Jeffrey e SOLOWAY, Elliot (1985) "Preprogramming Knowledge: A Major Source of Misconceptions in Novice Programmers", Human-Computer Interaction, vol.1, pp.133-161

BONAR, Jeffrey et al (1986) "Bridge: an Intelligent Tutor for Thinking About Programming", Proceedings of the ICAI Research Workshop, Windermere, Cumbria

BOULAY, Benedict Du, O'SHEA, Tim e MONK, John (1981) "The Black Box Inside the Glass Box: Presenting Computing Concepts to Novices", Int. J. Man-Machine Studies 14, 237-249

BOULAY, Benedict Du et al (1987) "Computers Teching Programming: An Introductory Survey of the Field", in Lawler, R. W. e Yazdani, M. (eds) "Artificial Intelligence and Education Volume One Learning Environments & Tutoring Systems" Ablex Publishing Corporation

CARVALHO, Sergio E. R. de (1982) "Introdução à Programação com Pascal", Editora Campus Ltda, Rio de Janeiro, RJ, Brasil

CHI, M. T. H. et al (1981) "Categorization and Representation of Physics Problems by Experts and Novices", Cognitive Science, 5, pp. 121-152

CLANCEY, William J. e SOLOWAY, Elliot (1990) "Artificial Intelligence and Learning Environments" Artificial Intelligence,vol 42, n.1, pp.1-6

CERRI, S. A. et al (1984) "The Little Lisper", AISB Quaterly Newsletter, 50

COLLINS, William J. (1986) "Intermediate Pascal Programming: A Case Study Approach", McGraw-Hill Book Company, New York, EUA

DETIENNE, Françoise e SOLOWAY, Elliot (1988) "An Empirically-Derived Control Strucyure for The Process of Program Understanding", Rapports de Recherche, n.886, Aout 1988, INRIA

DIJKSTRA, E. W. (1976) "A Discipline of Programming", Prentice-Hall, New Jersey

DILLENBOURG, P. et al "Training Transfer: A Bridge Betwen the Theory Orientede and Product-Oriented Approaches of ITS Design", TECFA Document 90-3, Université de Geneve, Faculté de Psychologie et des Sciences de l'Education

DILLENBOURG, P. "Design Human-Computer Collaborative Learning", TECFA Document 90-7, Université de Geneve Faculté de Psychologie et des Sciences de l'Education

diSESSA, Andrea A. e ABELSON, Harold (1986) "Boxer: A Reconstructible Computacinal Medium" Communications of the ACM, vol. 29, n.9, pp.859-868

EISENSTADT, Marc (1983) "A User-Friendly Software Environment for the Novice Programmer", Communications of the ACM, vol 26, n.12, pp.1058-1064

FINZER, William e GOULD, Laura (1984) "Programming by Rehearsal", Byte, june, pp.187-210

GENTNER, D.(1983a) " Structure-Mapping: A Theoretical Framework for Analogy, Cognitive Science, 7, pp. 155-170

GENTNER, Dedre e STEVENS, Albert L. (eds.) (1983) "Mental Models", Lawrence Erlbaum Associates, Publishers

GIVE'ON, Y. S. (1989) "Teaching Recursive Program Composition in Procedural Environments", Machine-Mediated Learning, volume 3, pp. 125-145

GOLDSTEIN, I. P. (1975) "Summary of Mycroft: a System for Understanding Simple Pictures Programs", Artificial Intelligence, 6, pp. 249-288

GREENO, James G. (1983) "Conceptual Entities" in Gentner, D. e Stevens, A. L. (eds) "Mental Models" Lawrence Erlbaum Associates, Publishers

GREENO, James G. (1988) "Situations, Mental Models, and Generative Knowledge", Institute for Research on Learning Report N. IRL88-0005, January

HARTLEY, J. R. (1973) "The Design an Evaluation of an Adaptative Teaching System", International Journal of Man-Machine Studies, 5, pp. 421-436

HARVEY, B. (1985) "Computer Science Logo Style", MIT Press, Cambridge, MA

HOARE, C. A. R., DAHL, O. J. e DIJKSTRA, E. W. (1972) "Structured Programming", Academic Press, New York

HOC, J. M. (1977) "Role of Mental Representation in Learning a Programming Language" International Journal of Man-Machine Studies 9, pp.87-105

JOHANSON, Roger P. (1988) "Compute, Cognition and Curriculum: Retrospect and Prospect", J. Educational Computing Research, vol 4, 1, pp.1-30

JOHNSON-LAIRD, P. N. (1980) "Mental Models in Cognitive Science", Cognitive Science, vol. 4, pp.71-115

JOHNSON, W. Lewis et al (1985) "Proust", Byte, 10, 4

JOHNSON, W. Lewis (1990) "Understandig and Debugging Novice Programs", Artificial Intelligence, 42, pp.51-97

KAHNEY, Hank (1989) "What do Novices Programmers Know About Recursion?" in Soloway, E. e Sphorer, J. C. (eds) "Studying the Novice Programmer", Lawrence Erlbaum Associates, Publishers

KELLER, A. M. (1982) "A First Course in Computer Programming Using Pascal", McGraw-Hill, Computer Science Series

KERNIGHAN, Brian W. (1984) "The UNIX Programming Environment" in Barstow, D. R. et al (eds) "Interactive Programming Environments", McGraw Hill Book Company, New York, EUA

KLEER, Johan de e BROWN, John Seely (1983) "Assumptions and Ambiguities in Mechanistic Mental Models" in Gentner, D. e Stevens, A. L. (eds) "Mental Models" Lawrence Erlbaum Associates, Publishers

KOFFMAN, E. B. et al (1975) "Artificial Intelligence and Automatic Programming in CAI", Artificial Intelligence, 6, pp. 215-234

KOFFMAN, E. B. (1985) "Problem Solving and Structured Programming in Pascal", Addison-Wesley

KOWALTOVSKI, Tomasz (1983) "Implementação de Linguagens de Programação", Editora Guanabara Dois, SP, Brasil

KUPER, Ron I. (1989) "Dependency-Directed Localization of Software Bugs", MIT Artificial Science Laboratoty, Technical Report 1053, may

LARKIN, Jill H. e SIMON, Herbert A. (1987) "Why a Diagram is (Sometimes) Worth Ten Thousand Words" Cognitive Science, 11, pp.65-69

LAWLER, Robert W. e YAZDANI, Masoud (ed.) (1987) "Artificial Intelligence and Education Volume One Learning Environments & Tutoring Systems" Ablex Publishing Corporation

LETOVSKY, S.(1986)"Cognitive Process in Programming Comprehension", in Soloway, E.(ed) "Empirical Studies of Programming", Norwood, N.J.: Ablex

LIEBERMAN, Henry (1984) "Seeing what your programs are doing", International Journal of Man-machine Studies, 21, 311-331

LIEBERMAN, Henry (1987) "An Example-Based Environment for Beginning Programmers", in Lawler, R. W. e Yazdani, M. (eds) "Artificial Intelligence and Education Volume One Learning Environments & Tutoring Systems", Ablex Publishing Corporation

MAYER, Richard E. (1979) "A Psychology of Learning BASIC" Communications of the ACM, vol.22, n.11

MAYER, Richard E. (1981) "The Psychology of How Novices Learn Computer Programming", Computing Surveys, Vol 13, n. 1

MAYER, Richard E. e BAYMAN, Piraye (1981) "Psychology of Calculator Languages: A Framework for Describing Differences in User's Knowledge", Communications of the ACM, vol.24, n.8, pp.511-520

MENDELSOHN, Patrick, GREEN, T.R.G, BRNA, Paul "Programming Languages in Education" TECFA Document 90-8, Université de Geneve, Faculté de Psychologie et des Sciences de l'Education

NEWELL, A. e SIMON, H. (1972) "Human Problem Solving", Englewood Cliffs, N. J.: Prentice-Hall

NORMAN, Donald A. (1983) "Some Observations in Mental Models", in Gentner, D. e Stevens, A. L. (eds) "Mental Models" Lawrence Erlbaum Associates, Publishers

PARK, W. B. (1990) "User-Centered Design", in Norman, D. (ed), "The Psychology of Everyday Things", Basic Books, Inc., Publishers New York, pp. 187-217

PIAGET, J. (1926) "A Representação do Mundo da Criança", Ed. Record, RJ, Brasil

PIAGET, J. (1970) "Piaget's Theory", in Mussen, P.H. (ed), "Carmichael's Handbook of Child Psychology", New York: John Wiley e Sons, pp. 703-732

PYLYSHYN, Zenon W. (1973) "What The Mind's Eye Tells The Mind's Brain: A Critique of Mental Imagery" Psychological Bulletin, vol.80, n.1

ROCHA, Heloísa V. (1989) "Ambiente Computacional Auxiliar no Processo de Aprender a Programar", Anais do Seminário de Informática na Educação: Um Desafio", IBM - Instituto de Engenharia de Software, Friburgo, RJ, dezembro

ROCHA, Heloisa V. (1990) "Representaçoes Computacionais Auxiliares no Aprendizado de Programação", Anais do Seminário de Informática Educativa, MEC/OEA/PRODEPA, Belém, agosto

ROCHA, Heloisa V. (1990) "Aprender a Programar: Por que Tão Difícil?", Anais da I Jornada Alagoana de Informática na Educação, UFAL, Novembro

SANDEWALL, E. (1984) "Programming in an Interactive Environment: The Lisp Experience" in in Barstow, D. R. et al (eds) "Interactive Programming Environments", McGraw Hill Book Company, New York

SAVITCH, Walter J. (1987) "Pascal: An Introduction to The Art and Science of Programming" The Benjamin/Cummings Publishing Company, Inc, Menlo Park, California, EUA, 2nd Edition

SCHIMITZ, Eber Assis et al (1985) "Pascal e Técnicas de Programação" Livros Técnicos e Cientificos Editora S.A, Rio de Janeiro, RJ, Brasil

SCHNEIDER, G. Michael et al (1978) "An Introduction to Programming and Problem Solving With Pascal", John Wiley e Sons, New York

SCHON, D. A. (1990) "The design Process", in Howard, V. A. (ed), "Varieties of Thinking: Essays from Harvards Philosophy of Education", Research Center, Nyc, Ny: Routledge, pp. 11-141

SEBESTA, Robert W. (1989) "Concepts of Programming Languages" The Benjamin Cummings Publishing Company, Inc.

SHAPIRO, E. Y. (1982) "Algorithmic Programming Debugging", MIT Press, Cambridge, MA

SHEIL, B. A. (1981) "The Psychological Study of Programming", Computing Surveys, vol.13, 3

SHEIL, B. A. (1984) "Power Tools for Programmers" in Barstow, D. R. et al (eds) "Interactive Programming Environments", McGraw Hill Book Company, New York

SLEEMAN, Derek (1986) "The Challenges of Teaching Computer Programming" Communications of ACM, vol. 29, n.9

SMITH, D.C. et al (1982) "Designing the Star User Interface" Byte, april, pp-242-282

SOLOWAY, Elliot et al (1982) "What do novices Know About Programming?", in Shneiderman, B. e Badre, A.(ed), "Directions in Human-Computer Interaction", Norwood, NJ: Ablex

SOLOWAY, Elliot e EHRLICH, Kate (1984) "Empirical Studies of Programming Knowledge" IEEE Transactions of Software and Engineering, vol. SE-10, n. 5

SOLOWAY, Elliot (1986) "Learnig to Program = Learning to Construct Mechanisms and Explanations" Communications of ACM, vol. 29, n.9, pp.850-858

SOLOWAY, Elliot e SPOHRER, James C. (ed.) (1989) "Studyng the Novice Programmer" Lawrence Erlbaum Associates, Publishers

TAN, Yang Meng (1989) "A Program Design Assistant" MIT Artificial Intelligence Laboratory, A.I. Working Paper n.327, june

TEITELBAUM, Tim e REPS, Thomas (1981) "The Cornell Program Synthesizer: A Syntax-Directed Programming Environment" Communications of the ACM, vol. 24, n.9

TEITELMAN, W. e MASINTER, L. (1984) "The Interlisp Programming Environment", in Barstow, D. R. et al (eds) "Interactive Programming Environments", McGraw Hill Book Company, New York, EUA

TINKER, R. (1985) "Thinker's Toy", Hands On!, vol.7,1

TURKLE, S. (1984) "The Second Self: Computers and the Human Spirit", Simon and Schuster, New York

VALENTE, Ann Berger (1987) "Como o Computador é Dominado Pelo Adulto" Núcleo de Informática Aplicada à Educação - Memo 7 - UNICAMP

VALENTE, Ann Berger (1988a) "Aprender Programação Logo: Será que algumas pessoas tem mais facilidade que outras" Núcleo de Informática Aplicada à Educação - Memo 16 - UNICAMP

VALENTE, José Armando e VALENTE, Ann Berger (1988b) "LOGO: Conceitos e Aplicações" McGraw-Hill, SãoPaulo, SP, Brasil

VALENTE, José Armando (1989) "Estudo da Relação Entre Estilo de Programação Logo e Estilo Cognitivo" Núcleo de Informática Aplicada à Educação - Memo 10 - UNICAMP

WERTHEIMER, M. "Productive Thinking", (1959) Harper and Row, New York

WHITE, Barbara Y. (1981) "Design Computer Games to Facilitate Learning" MIT Artificial Intelligence Laboratory, Technical Report 619, february

WHITE, Barbara Y. e FREDERIKSEN, John R. (1990) "Causal Model Progressions as a Foundation for Intelligent Learning Environments" Artificial Intelligence, 42, pp. 99-157

WILANDER, J. (1984) "An Interactive Programming System for Pascal" in Barstow, D. R. et al (eds) "Interactive Programming Environments", McGraw Hill Book Company, New York, EUA

WINOGRAD, Terry e FLORES, Fernando (1987) "Understanding Computers and Thinking" Addison-Wesley Publishing Company, Inc.

YOUNG, Richard M.(1981) "The Machine Inside the Machine: User's Models of Pocket Calculators" Int. J. Man-Machine Studies, 15, pp.51-85

YOUNG, Richard M.(1983) "Surrogates and Mappings: Two Kinds of Conceptual Models for Interactive Devices" in Gentner, D. e Stevens, A. L. (eds) "Mental Models" Lawrence Erlbaum Associates, Publishers

Apêndice

SUBCONJUNTO DE INSTRUÇÕES IMPLEMENTADO

A seguir são apresentados em ordem alfabética os comandos implementados, juntamente com as respectivas definições de sintaxe e semântica. Ao lado de cada instrução aparece sua forma mnemônica, sendo que ambas são permitidas na implementação.

Quanto à notação utilizada:

<predi> indica uma operação do tipo predicado, isto é, cujo resultado é verdadeiro ou falso.

<vari> indica o nome de uma variável

<objeto> indica um número, palavra, lista ou o valor de uma variável que contém um desses objetos. Em alguns instrução existem restrições quanto ao tipo do objeto, que são ressaltadas na semântica da instrução.

1. APRENDA (AP)

sintaxe:
aprenda, ap <nome_do_procedimento> <:var1> <:var2> ... <:varn>
significado:
indica inicio da definição de um procedimento com n (>=0) parâmetros

2. ALGUM

sintaxe:
algum <pred1> <pred2> ou (algum <pred1> <pred2>....<predn>), com n>=2

significado:
retorna "verd sempre que um dos argumentos ou todos forem verdadeiros e "falso quando forem falsos todos os argumentos (operação lógica OR).

3. CARE

sintaxe:
care
significado:
Operação de entrada. Lê e retorna um caracter teclado.

4. COLOQUE (COL)

sintaxe:
coloque, col <objeto> <nome>
significado:
Comando de atribuição. Atribui à variável de nome <nome> o valor de <objeto>.

5. E

sintaxe:
e <pred1> <pred2> ou (e <pred1> <pred2>....<predn>), com n>=2
significado:
retorna "verd sempre que todos seus argumentos sejam verdadeiros e "falso quando qualquer um deles for falso (operação lógica AND).

6. ELEMENTO (ELEM)

sintaxe:
elemento, elem <número> <objeto>
significado:
retorna o elemento de objeto cuja posição dentro de <objeto> corresponda a <número>.

7. ENVIE

sintaxe:
envie <objeto>
significado:
termina a execução do procedimento e retorna <objeto> para o nivel imediatamente anterior.

8. ESCREVA (ESC)

sintaxe:
escreva, esc <objeto>
significado:
mostra <objeto> na tela, e muda o cursor para a linha seguinte

9. ÉLISTA

sintaxe:
élista <objeto>
significado:
Predicado que retorna "verd se <objeto> é uma lista, e "falso caso contrário.

10. ÉNUMERO

sintaxe:
énúmero <objeto>
significado:
Predicado que retorna "verd se <objeto> é um número, e "falso caso contrário.

11. ÉPALAVRA

sintaxe:
épalavra <objeto>
significado:
Predicado que retorna "verd se <objeto> é uma palavra, e "falso caso contrário.

12. ÉVAZIA

sintaxe:
évazia <objeto>
significado:
Predicado que retorna "verd se <objeto> é uma lista ou palavra vazia, e "falso caso contrário.

13. FIM

sintaxe:
fim
significado:
termina a definição de um procedimento

14. JUNTENOFIM (JF)

sintaxe:
juntenofim, jf <objeto> <lista>
significado:
acrescenta <objeto> à <lista>, como último elemento

15. JUNTENOINICIO (JI)

sintaxe:
juntenoinicio, ji <objeto> <lista>
significado:
acrescenta <objeto> à <lista>, como primeiro elemento

16. LINE

sintaxe:
line
significado:
Operação de entrada. Lê do teclado uma linha e retorna a lista formada pelo que foi teclado.

17. LISTA

sintaxe:
lista <objeto1> <objeto2> ou
(lista <objeto1> <objeto2>..<objeton>), com n>=2
significado:
retorna uma lista formada pelos seus argumentos. Junta as entradas criando uma única lista.

18. NÃO

sintaxe:
não <pred>
significado:
retorna "verd quando <pred> é falso e "falso quando <pred> é verdadeiro (operação lógica NOT).

19. NEL

sintaxe:
nel <objeto>
significado:
retorna o número de elementos de <objeto>

20. PALAVRA (PAL)

sintaxe:
palavra, pal <palavra1> <palavra2> ou (palavra, pal <palavra1> <palavra2>...<palavran>), com n>=2
significado:
retorna uma palavra composta das palavras dadas como argumento, na ordem em que são dadas.

21. PARE

sintaxe:
pare
significado:
indica pontos de término de execução de um procedimento.

22. PRIMEIRO (PRI)

sintaxe:
primeiro, pri <objeto>
significado:
retorna o primeiro elemento de <objeto>

23. REPITA

sintaxe:
repita <número> <lista de instruções>
significado:
Comando de repetição incondicional. Executa <lista de instruções> <número> vezes.

24. SÃOIGUAIS

sintaxe:
sãoiguais <objeto1> <objeto2> ou
(sãoiguais <objeto1> <objeto2>...<objeton>), com n>=2
significado:
retorna "verd sempre que seus argumentos forem iguais, e falso caso contrário

25. SE

sintaxe:
se <pred> <lista de instruções> ou
se <pred> <lista1 de instr.> <lista2 de instr.>
significado:
Foi implementada sómente a primeira forma, correspondendo ao comando - se ...então... - . Se <pred> é verdadeiro executa <lista de instruções>, caso contrário executa próximo comando. A segunda forma corresponde ao - se...então...senão... -. Se <pred> é verdadeiro executa <lista1 de instr.>, caso contrário executa <lista2 de instr.>.

26. SEMPRIMEIRO (SP)

sintaxe:
semprimeiro, sp <objeto>
significado:
retorna <objeto> sem o primeiro elemento.

27. SEMULTIMO (SU)

sintaxe:
semúltimo, su <objeto>
significado:
retorna <objeto> sem o último elemento.

28. ULTIMO (ULT)

sintaxe:
último, ult <objeto>
significado:
retorna o último elemento de <objeto>

29. OPERADORES RELACIONAIS INFIXOS

= igualdade
< menor que
> maior que

30. OPERADORES ARITMÉTICOS INFIXOS

+ soma
- subtração
* multiplicação
/ divisão