UNIVERSIDADE ESTADUAL DE CAMPINAS
FACULDADE DE ENGENHARIA ELÉTRICA E DE COMPUTAÇÃO

TESE DE DOUTORADO

Uma Ferramenta para Auxiliar no Ensino de Estruturas de Dados como Tipo de Dado Abstrato

AUTORA: Angela de Mendonça Engelbrecht
ORIENTADORA: Profa. Dra. Beatriz Mascia Daltrini

BANCA EXAMINADORA:

Prof. Dr. Luís Carlos Trevelin – UFSCAR
Profa. Dra. Suely Aparecida Galli Soares – PUC-Campinas
Profa. Dra. Heloísa Vieira da Rocha – UNICAMP
Prof. Dr. Mário Jino – UNICAMP
Prof. Dr. José Raimundo de Oliveira - UNICAMP

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

Fevereiro de 2003

FICHA CATALOGRÁFICA ELABORADA PELA
BIBLIOTECA DA ÁREA DE ENGENHARIA - BAE - UNICAMP

En32u

Engelbrecht, Angela de Mendonça
Uma ferramenta para auxiliar no ensino de estruturas de dados como tipo de dado abstrato / Angela de Mendonça Engelbrecht.-- Campinas, SP: [s.n.], 2003.

Orientadora: Beatriz Mascia Daltrini.
Tese (doutorado) - Universidade Estadual de Campinas, Faculdade de Engenharia Elétrica e de Computação.

1.Estruturas de dados (Computação). 2. Tipos abstratos de dados (Computação). 3.Ensino auxiliado por computador. I. Daltrini, Beatriz Mascia. II. Universidade Estadual de Campinas. Faculdade de Engenharia Elétrica e de Computação. III. Título.

Ao meu marido Walter e aos meus filhos Raquel e Rafael as maiores alegrias da minha vida.

RESUMO

A importância de se ensinar e aplicar o conceito de abstração em programação desde as séries iniciais dos cursos de ciência da computação e a contribuição que os recursos visuais oferecem para auxiliar no ensino levaram à construção da ferramenta ADTTool. Este trabalho apresenta ADTTool (Abstract Data Type Tool), uma ferramenta desenvolvida para auxiliar no ensino das estruturas de dados. Ela foi desenvolvida considerando-se a importância de ensinar as estruturas de dados como tipos de dados abstratos (TDA). A ferramenta permite que o estudante teste isoladamente cada uma de suas funções, que usam as operações da estrutura de dados, sem que estas tenham sido implementadas. A ferramenta possui uma interface gráfica que faz a animação da função através da demonstração do fluxo dos dados entre as variáveis e estruturas de dados que ela possui. A interface retrata as representações criadas para as estruturas de dados de modo independente da forma como foram armazenadas, mantendo a abstração, e permitindo que os estudantes identifiquem mais facilmente os elementos de sua função, durante a demonstração.

ABSTRACT

The importance of teaching and applying abstract concepts in programming early in a Computer Science course, plus the contribution to learning provided by visual resources, led to the development of the ADTTool. This dissertation presents ADTTool (Abstract Data Type Tool), a software tool developed to support teaching data structure. ADTTool was developed considering the importance of teaching data structures as abstract data types (ADT). Using the ADTTool the students can test separately each function, making use of data structures operations without implementing these data structures or operations. The ADTTool has a graphic interface which animates the function with data flow demonstration of its variables and data structures. The interface shows the data structure's representation independently from the way it has been stored, keeping it abstract and allowing the students to identify their function elements more easily, during demonstration.

Agradecimentos

Inicialmente, gostaria de agradecer à minha orientadora Profa. Beatriz Mascia Daltrini pelo constante incentivo e por sempre acreditar no meu trabalho, prontificando-se a discutir quaisquer aspectos deste projeto, não importando o dia e a hora, mesmo sendo em suas horas de descanso e de dedicação à família.

Aos Professores João Luís Garcia Rosa, José Estêvão Picarelli, José Oscar Fontanini de Carvalho e Orandi Mina Falsarella pelas sugestões e o constante encorajamento durante todo o trabalho.

A todos os demais colegas das Faculdades de Engenharia de Computação e de Análise de Sistemas, da Pontifícia Universidade Católica de Campinas pelo apoio e pela torcida.

À Pontifícia Universidade Católica de Campinas por proporcionar a capacitação de seus docentes.

Aos meus alunos Bruno, Fernanda, Juliana, Rafael, Rodrigo e Thiago por participarem voluntariamente do processo piloto de apreciação da ferramenta e por seus comentários.

SUMÁRIO

LISTA DE FIGURAS

INTRODUÇÃO

Face ao surgimento do paradigma de orientação a objetos, utilizado nas mais modernas tecnologias para o desenvolvimento de software, tornou-se necessária a revisão na forma de ensinar estruturas de dados. A abstração dos dados e dos processos fundamenta esse paradigma e constitui uma forma diferente de pensar o projeto de um sistema de software. Assim sendo, surgiu a necessidade de empregar o conceito de abstração desde as primeiras séries dos cursos da área de computação.

Ensinar estruturas de dados não é simplesmente mostrar para o estudante a ação das operações sobre as informações que a estrutura contém. Ensinar estruturas de dados é ajudar o estudante a desenvolver sua lógica de construção de programas, em problemas que usam as estruturas de dados como forma de armazenamento das informações ou como estruturas auxiliares na solução de problemas. A pilha, por exemplo, é uma estrutura de dados que deve ser compreendida não só pela sua característica de possuir ações que permitam a inserção e a eliminação de informações do seu topo, mas também pelo seu uso na solução de problemas, como a avaliação de expressões aritméticas.

Duas abordagens podem ser adotadas ao ensinar estruturas de dados. Uma delas consiste em apresentar ao estudante a implementação das operações de uma estrutura de dados, juntamente com sua definição. Isto significa escolher o meio de armazenamento da estrutura: se em representação contígua de memória (arranjos) ou em representação ligada (estática ou dinâmica). Esta abordagem gera dificuldades relacionadas à compreensão, por parte do estudante, que não se constitui boa prática manipular diretamente a informação armazenada na estrutura de dados, mas sim, através das operações associadas à estrutura de dados.

Outro problema associado a esta abordagem se refere ao adiamento do ensino do uso de técnicas de divisão do problema em módulos; a manipulação direta da informação armazenada não favorece a criação de módulos que implementam operações nesta estrutura. A conseqüência desta abordagem é a geração de vícios de programação não modular que dificultam o aprendizado do paradigma de orientação a objetos e do processo de abstração envolvido no desenvolvimento de um software.

Em uma outra abordagem, as operações de uma estrutura de dados não são implementadas a medida que são definidas. As operações são apresentadas aos estudantes através das modificações que elas provocam, quando são executadas, independentemente do meio que serão armazenadas. Neste caso, a importância maior da operação é destacada para a ação que ela provoca e não para a sua representação interna. Desta forma, passa-se a trabalhar com as estruturas de dados como tipos de dados abstratos (TDA). Esta abordagem é mais difícil, inicialmente, de ser compreendida pelos estudantes, pois causa um grau de resistência na utilização das operações da estrutura de dados, uma vez que o local de armazenamento das informações não é visível. Apesar da dificuldade inicial, esta abordagem atua como o elemento facilitador na utilização posterior do paradigma de orientação a objetos, contribuindo para a compreensão dos níveis de abstração existentes na construção de um software.

O significado de abstração neste contexto está diretamente associado ao tipo de dado, significando que a utilização das operações definidas para este tipo de dado deve ser feita independentemente do meio que será escolhido para o armazenamento dos dados.

Há muito tempo a literatura traz o conceito de tipos de dados abstratos (TDA). No entanto, a partir do momento em que se empregam as estruturas de dados na solução de problemas, a abstração é deixada de lado, pois a estrutura de dado é então apresentada em uma das formas de representação interna: contigüidade física ou ligada. A partir do momento em que se determinou onde a estrutura será armazenada, o conceito de tipo de dado abstrato não está mais sendo empregado. Em se tratando de estudantes de séries iniciais, trabalhar

diretamente com a representação interna de uma estrutura de dados é mais simples, por se tratar do tipo concreto estabelecido; porém, o conceito de tipo de dado abstrato é perdido.

Ensinar, então, estruturas de dados como tipos de dados abstratos resultou em duas necessidades. A primeira delas refere-se à necessidade de se estabelecer uma forma de trabalhar com as operações das estruturas de dados, sem definir o espaço de armazenamento, isto é, buscar uma especificação formal para as estruturas de dados. A segunda necessidade surgiu com a importância de exercitar a aplicação de estruturas de dados, pela execução de funções construídas pelo estudante, ainda que neste momento, a abstração criada sobre as operações seja mantida.

Para o atendimento da primeira necessidade é proposta uma nova definição para tipo de dado abstrato através da inclusão de uma representação visual, que foi determinada para cada uma das estruturas de dados.

Para o atendimento da segunda necessidade foi criada uma ferramenta de apoio à execução de funções que utilizam operações abstratas sobre as estruturas de dados. Em um ambiente de programação, para a execução da função é necessária a construção de funções que representam as operações sobre as estruturas de dados. Isto significa escolher onde armazenar os dados da estrutura de dados. A partir do momento em que o estudante escolhe a representação interna da estrutura de dados ele passa a pensar no concreto.

A ferramenta Abstract Data Type Tool - ADTTool foi então criada, permitindo que funções que utilizam estruturas de dados sejam implementadas e executadas antes do estudante passar para a fase de escolha da representação interna da estrutura. Com isto ele deverá corrigir os erros de sua função utilizando apenas as operações da estrutura de dados, concentrando-se tão somente na funcionalidade do algoritmo construído.

Além de permitir que uma função isolada seja executada, a ADTTool oferece uma interface gráfica que faz a animação da função, através da demonstração do fluxo de dados entre as estruturas e variáveis simples que a

função possui. O emprego da animação na ferramenta se deve à importância que os recursos visuais trazem para os ambientes ligados ao aprendizado.

A ferramenta permite ainda, a visualização passo a passo do algoritmo, onde o estudante poderá acompanhar o desempenho da função, permitindo-lhe uma melhor compreensão da ação das operações sobre as estruturas utilizadas; ele poderá visualizar o passo onde um determinado erro ocorre ou onde foi feito o uso de uma ação não-desejada.

Apresentação da tese

No Capítulo I, são apresentadas algumas considerações sobre o conceito de abstração empregado em vários segmentos da computação e alguns destaques para a importância de sua abordagem nas séries iniciais do curso. Essas considerações sobre abstração formam o embasamento teórico para a premissa de se ensinar estruturas de dados como tipos de dados abstratos.

Considerando-se que a ferramenta faz a animação de uma função, o Capítulo II apresenta algumas considerações sobre a influência dos recursos visuais e do computador no ensino. Apresenta também algumas ferramentas e ambientes de animação existentes.

Alguns aspectos sobre a usabilidade de um software são apresentados no Capítulo III, destacando aqueles que foram observados para a construção da ferramenta, principalmente no que se refere a interfaces gráficas. São destacados também alguns ambientes para a criação de interfaces gráficas.

O Capítulo IV apresenta a formalização das estruturas de dados: listas lineares, pilhas e filas como tipos de dados abstrato. Nesta formalização destaca-se a proposta de uma nova definição para o tipo de dado abstrato, através da inclusão da representação visual criada para cada uma das estruturas de dados. Apresenta também as ações através de especificação formal, caracterizando a abstração necessária para o uso das operações nas funções criadas pelos estudantes.

A descrição da ferramenta ADTTool, relativamente aos componentes que possui e aos elementos da interface, é apresentada no Capítulo V.

O Capítulo VI apresenta como utilizar a ferramenta ADTTool, e as possíveis seqüências válidas de escolhas dos botões que a interface possui, que auxiliam na descrição de seu uso.

O Capítulo VII apresenta a metodologia utilizada no experimento feito com um grupo de alunos voluntários para a demonstração da ferramenta.

A conclusão apresenta as contribuições deste trabalho, sugestões para a próxima versão da ferramenta e algumas considerações sobre continuidade da pesquisa.

O Anexo I apresenta um conjunto classificado e comentado de ferramentas levantadas durante este trabalho.

O Anexo II apresenta alguns conceitos básicos empregados na criação de um grafo de estados relativo aos botões utilizados na interface da ferramenta.

O Anexo III apresenta um guia para a utilização do disco que acompanha esta tese.

CAPÍTULO I

CONSIDERAÇÕES SOBRE O CONCEITO DE ABSTRAÇÃO EM PROGRAMAÇÃO

Introdução

São feitas aqui algumas considerações sobre o conceito de abstração sob a ótica da ciência da computação e o seu emprego em várias fases empregadas para a construção de um software. Há um destaque para o envolvimento deste conceito com o paradigma de orientação a objetos e a importância de introduzi-lo nas séries iniciais ao se trabalhar com a criação de definição de novos tipos de dados.

1.1 – Considerações Iniciais

O computador é uma máquina abstrata. Independentemente de conhecer como ele executa as tarefas, interessa saber apenas o que ele pode executar.

A abstração é um conceito antigo usado inicialmente para processos. SIMULA 67 foi a primeira linguagem com recursos para a abstração de dados. Barbara Liskov e Stephen Zilles [LIS74], em 1974, definiram o conceito de tipo de dado abstrato como uma classe de objetos abstratos. Liskov e Zilles afirmam que "... o que desejamos de uma abstração é um mecanismo que permita a expressão de detalhes relevantes e a supressão de detalhes irrelevantes". Destacam, ainda,

que a abstração é um termo usado em vários segmentos e por várias pessoas como a chave para a construção de um bom projeto.

Na engenharia de software, ao se adotar a divisão do problema em módulos estabelecem-se níveis de abstração. No mais alto nível de abstração, os termos usados para a declaração são os do próprio ambiente do problema. O mais baixo nível de abstração está mais próximo dos detalhes de implementação [PRE92].

A divisão de um problema maior em módulos não deve ser aleatória e a arquitetura do software gerada pode variar de acordo com diversidade de tendências, estilos e experiência de quem o projeta. Entretanto, há técnicas que auxiliam na construção dessa divisão de forma que o resultado final contemple requisitos de um bom projeto, tais como a busca de módulos com alta coesão e baixo acoplamento.

A coesão e o acoplamento existentes entre os módulos divididos afetam o nível de abstração de um projeto. A abstração é a arte de expor seletivamente a funcionalidade de interfaces coesivas ao mesmo tempo em que esconde as implementações acopladas [SCH97].

1.2 - Considerações sobre abstração nas séries iniciais

Dominar a habilidade de pensar abstratamente sobre os componentes de um programa de computador é problemático para os estudantes de séries iniciais de cursos de computação. O estudante, enquanto não desenvolve esta habilidade, tende a ver um programa como uma coleção não-estruturada de declarações e expressões. Com isso, a complexidade de um programa aumenta com o tamanho do problema a ser desenvolvido. Ao atingir a habilidade de pensar abstratamente, o estudante tende a ver o programa como uma coleção de funções e classes [TUR01].

Antony [ANT00] destaca que a abstração que envolve os dados encapsulados, também chamados de tipos de dados abstratos, representa o conceito de linguagem de programação mais promissor para apoiar a engenharia

de software. Destaca, ainda, que o tipo de dado abstrato é a base para a filosofia de projetos com informações ocultas, o que torna o software com maior facilidade de entendimento e análise, e que também permite uma melhor manutenção e reutilização.

A abstração e a informação oculta apresentam-se como princípios fundamentais e essenciais no desenvolvimento do software e se destacam como elementos importantes para serem ensinados nas primeiras séries de um curso da área da ciência da computação. Esses princípios podem ser ensinados como elementos que completam ou fazem parte de uma idéia geral. Para isso devem-se ocultar detalhes de um sistema complexo, enquanto é fornecido um elemento substituto mais simples para explicar os aspectos mais importantes do problema [BUC01].

A compreensão do conceito de abstração é fundamental, pois representa a base para a tecnologia de orientação a objetos. Um ponto favorável a essa tecnologia é o fato de que ela é mais estruturada e modular do que a programação orientada a função, produzindo programas nos quais é mais fácil fazer manutenção e em que se permite a reutilização [SEN94].

Programadores têm adotado a abordagem de orientação a objetos para melhorar a abstração dos dados, a modularidade e a reutilização de código. Essas características podem potencialmente beneficiar tanto o processo de desenvolvimento de software quanto a qualidade do software gerado [STA94].

Bucci [BUC01] destaca a preocupação com o ensino de elementos abstratos e a ligação deles com o concreto, dizendo que, quando se ensina objetos, usando-se classes abstratas em C++ ou interfaces em Java, junto com a classe concreta, não se está ensinando informações ocultas e abstração. As linguagens de programação oferecem mecanismos para ocultar informações; porém a compreensão humana sobre a informação oculta fica comprometida, quando é apresentada aos estudantes a construção da classe concreta em continuidade à apresentação de classe abstrata.

1.3 – Comentários Finais

O uso do conceito de abstração nas séries iniciais é importante na medida em que permitirá que o estudante comece desde cedo a praticar os conceitos de engenharia de software.

As listas lineares, pilhas e filas são estruturas de dados que, devido ao conjunto de operações que definem claramente a ação sobre os dados que elas possuem, representam elementos adequados para a introdução do conceito de abstração.

Alguns conceitos de abstração estão fortemente ligados ao paradigma de orientação a objetos. Mesmo não introduzindo os conceitos deste paradigma, que é assunto de outra disciplina, é possível trabalhar com os recursos de uma linguagem de programação de forma que se empregue a abstração no aprendizado das estruturas de dados, tratando-as como tipos de dados abstratos.

Assim sendo, ao não empregar os conceitos de classes e objetos do paradigma de orientação a objetos, as classes concretas mencionadas por Bucci [BUC01] referem-se à construção das operações das estruturas de dados e conseqüentemente à definição do local de armazenamento das operações. E as classes abstratas referem-se ao uso das estruturas de dados como tipo de dado abstrato.

Portanto, para uma boa compreensão do uso da abstração no ensino de estruturas de dados é necessário trabalhar com estruturas como de tipos de dados abstratos antes de construir as operações propriamente ditas que elas possuem.

CAPÍTULO II

CONSIDERAÇÕES SOBRE RECURSOS VISUAIS NA COMUNICAÇÃO ENTRE O HOMEM E O COMPUTADOR

Introdução

Neste capítulo faz-se uma apresentação sobre alguns recursos visuais que têm sido desenvolvidos e usados para facilitar a comunicação entre o homem e o computador, mais especificamente aqueles envolvidos com o processo de aprendizagem de estudantes de cursos de computação.

2.1 - Considerações Iniciais

Compreender as estruturas de dados como tipos de dados abstratos, na sua essência, significa praticar o uso das estruturas na solução de problemas, porém sem implementar as operações definidas para elas. Isso significa incentivar o estudante a trabalhar a sua habilidade de pensar abstratamente o problema a ser resolvido.

Usar as estruturas de dados sem ter implementado suas operações, buscando compreender a abstração desse tipo de dado, é possível por meio de um ambiente de execução. Este ambiente deve estar preparado para interpretar as funções criadas pelo estudante para a solução do problema, onde é possível validar a funcionalidade desejada para a aplicação. Além disso, é necessário, dar

condições ao estudante para verificar a aplicabilidade da abstração associada à estrutura de dado utilizada na solução do problema.

Um ambiente para executar as funções poderia simplesmente embutir a definição das estruturas de dados de forma que o estudante pudesse fazer uso das operações sem saber como foram implementadas; a seguir poderia interpretar a função criada pelo estudante e finalmente apresentar um resultado, como uma saída normal de console.

Com a integração de recursos visuais ao ambiente, é possível melhorar a comunicação, por exemplo, demonstrando na tela, passo a passo, a função construída pelo estudante, permitindo-lhe identificar a ação executada naquele momento e verificando se atende à funcionalidade desejada para a função.

O uso do computador como componente dos ambientes de ensino tem aumentado em parte pelas facilidades que os avanços tecnológicos têm trazido, em termos de velocidade de processamento, recursos de comunicação e capacidade de armazenamento. Com esses recursos tem sido possível criar aplicações cada vez mais sofisticadas com elementos que podem auxiliar no aprendizado em todas as áreas de ensino. Além disso, o uso de figuras e recursos visuais há muito tempo vêm sendo utilizado na prática como auxiliares no processo educacional através da inclusão de gráficos e figuras.

2.2 - Considerações sobre o Uso do Computador e de Representações Visuais no Ensino

O uso do computador e das tecnologias de comunicação como componentes auxiliares nos processos de ensino tem aumentado muito nos últimos tempos.

Em 1978, Towle e DeFanti [TOW78], com o objetivo de criarem uma ferramenta para auxiliar no ensino de computação gráfica e de programação de jogos para ser executada pelos estudantes em suas residências, desenvolveram um programa interativo chamado Gain (Grass Assisted Instruction), cujo objetivo era auxiliar no ensino da linguagem GRASS, voltada à computação gráfica. Isso

ocorreu em uma época cuja tendência era a de aquisição de computadores para uso em residências, os computadores pessoais. O objetivo era o de ensinar os conceitos fundamentais da linguagem, em relação ao uso de variáveis, comandos iterativos e condicionais.

A literatura apresenta vários termos que classificam as diversas formas de ensino usando esses recursos, tais como CAL (computer-aided learning), CBT (computer-based training).

Schär [SCH00] apresenta CAL como uma forma diferente de ensinar, na qual o estudante tem o computador como um professor virtual e diz que o desenvolvimento de ferramentas para apoiar o ensino tem sido um desafio se comparado aos métodos tradicionais, pois deve haver o envolvimento de várias áreas ligadas ao ensino. Schär destaca vários aspectos envolvidos nesta forma de ensinar que devem ser observados: aspectos pedagógicos, mídia, tecnologias disponíveis, modelos, aceitação pelo usuário.

Os aspectos pedagógicos ligados à teoria do aprendizado devem ser adequados a esse novo paradigma. A escolha da mídia deve considerar as particularidades que cada tipo possui para transmitir uma informação e o nível de receptividade que cada uma delas produz em cada um dos estudantes. Por exemplo, o resultado obtido pela transmissão de uma determinada informação utilizando-se um recurso sonoro é diferente daquele obtido se utilizado para a transmissão um recurso visual ou a combinação de ambos os recursos. A escolha da estratégia de ensino a ser empregada (sistemas do tipo simulação, sistemas tutor, hipertexto ou hipermídia) deve estar em conformidade com as tecnologias disponíveis, incluindo os sistemas. Os modelos de informação devem ser empregados na comunicação entre o homem e o computador, pois as diversas formas de interagir com o computador influenciam o processo cognitivo do aprendizado. Por exemplo, o emprego de mecanismos de “feedback” possibilita ao estudante aprender com os próprios erros. A preocupação com a aceitação desse novo paradigma por parte do usuário deve levar em consideração, entre outras coisas, a facilidade ou não do manuseio da ferramenta e a disponibilidade de acesso ao ambiente onde a ferramenta está.

Um sistema de simulação interativa, como um exemplo de CAL, fornece um ambiente rico que pode ser explorado livremente pelo usuário. É uma ferramenta de aprendizado flexível. A partir de um ponto de vista pedagógico, atende a uma filosofia construtivista de aprendizado. Em uma simulação interativa, o estudante pode explorar interativamente as condições de ações e eventos do ambiente, formulando suas próprias hipóteses e comparando sua intuição com o resultado obtido [SCH00].

Um sistema de treinamento baseado no computador (CBT) é um sistema tutor que estabelece um diálogo com o estudante através de passos pré-programados. Cada passo deve ter uma meta a ser cumprida representando um objetivo diferente de aprendizado. Estes passos devem possuir uma ligação hierárquica, com graus de dificuldade diferenciados e crescentes. O sistema deve testar com freqüência se o estudante atingiu o objetivo esperado para prosseguir para uma nova etapa. Nesse tipo de sistema o estudante não exerce controle sobre os procedimentos e deve ser orientado quanto à seqüência dos passos seguintes.

Prechelt [PRE01] e Humphrey [HUM96] abordam o processo de aprendizado em programação a partir da experiência, o que permite que se realize uma análise dos erros praticados e posterior correção.

Humphrey desenvolveu um processo completo de aprendizado individual chamado PSP – Personal Software Process, que é composto de um conjunto de passos bem definidos dentro de um processo de desenvolvimento de software que envolve projeto, codificação, compilação e teste. Em cada passo o indivíduo aprende a coletar e estimar medidas relativas a tamanho e tempo, envolvidos em cada fase do processo, além de registrar problemas enfrentados, erros encontrados e cometidos. Com esses registros é possível criar uma lista de itens de verificação, para ser utilizada na fase de revisão do projeto e da codificação. A partir da observação dos itens de verificação o usuário aprende a avaliar seu programa, aumentando o seu desempenho pessoal.

Prechelt, considerando como ponto negativo o tempo necessário para o aprendizado da metodologia de PSP, desenvolveu uma técnica mais simples,

porém baseada na mesma idéia, chamada DLDA – Defect Logging and Defect Data Analysis. Consiste ela de apenas duas etapas, como o próprio nome indica: a fase de registro dos defeitos encontrados durante todo o processo de construção de um programa e a segunda fase de análise dos dados coletados, tais como, números de compilações necessárias, número de linhas do programa, o tempo total gasto. Os dados coletados sobre erros cometidos não são necessariamente de um programa, podem ser acumulados e analisados posteriormente, e é possível analisar a ocorrência de um mesmo erro em mais de um programa. Na análise dos dados, os erros coletados são classificados de acordo com determinadas categorias. A coleta dos dados é automática bem como a tabulação de alguns erros; por exemplo, por fase, por tipo de erro.

Os recursos audiovisuais são envolvidos como componentes adicionais do processo de aprendizagem, pois permitem registrar e demonstrar elementos da realidade muitas vezes inatingíveis pelo senso comum, como movimentos tridimensionais, funcionamento e interior de objetos, além de promover a concentração e chamar a atenção sobre o objeto sendo observado [LIM99].

Pilgrim [PIL00] declara que os benefícios obtidos com o uso adequado de elementos visuais não podem ser ignorados, uma vez que melhoram a exatidão e plenitude com as quais os usuários atingem o objetivo especificado.

A estrutura visual como forma de representação ajuda na interpretação de uma idéia ou de um pensamento. Sem entrar no mérito de aspectos pedagógicos, em determinadas situações os gráficos, figuras e imagens são elementos superiores se comparados às palavras [NOR94].

Assim, com os recursos computacionais disponíveis e as facilidades que as representações visuais proporcionam ao processo de aprendizagem, a animação de uma função, através da demonstração visual do movimento dos dados entre as variáveis pertencentes a ela, propicia ao estudante a possibilidade de acompanhar passo a passo a execução da função. Com o acompanhamento, o estudante pode determinar os possíveis pontos dentro da função que não atendem à funcionalidade desejada, compreender a ação de cada instrução que utilizou e, principalmente, constatar que é possível usar um conjunto único de operações de

uma estrutura de dados, para resolver vários problemas, sem dar grande importância como foram implementadas estas operações.

2.3 – Considerações sobre Linguagens Visuais

Uma linguagem visual é usada como uma das formas para a comunicação entre homem e o computador. Os elementos que a compõem são basicamente objetos gráficos que podem ser organizados em duas ou três dimensões.

A descrição formal da maioria das linguagens visuais é composta de um vocabulário, representado pelos objetos gráficos, relações entre esses objetos e um conjunto de regras. Os objetos gráficos possuem atributos que os distinguem entre si [COS97].

A grande vantagem das linguagens visuais é a capacidade que possuem de fornecer aos humanos uma comunicação e uma interação com o computador através de representação visual apresentada na tela do computador [NAR98].

Narayanan [NAR98] apresenta uma caracterização sobre as formas de apresentação visual na comunicação entre o homem e o computador. Uma mesma ferramenta de aplicação visual pode possuir mais de uma dessas características. Entre elas, define: representações visuais, linguagem visual, programação visual, linguagem de programação visual, visualização de software e animação de algoritmo.

A representação visual é definida como sendo a forma utilizada para expressar uma informação, tais como, diagramas, figuras geométricas, ícones ou outros elementos, de tal modo a levar o usuário a fazer inferências sobre o que estas formas estão representando.

A linguagem visual é toda aquela que possui em seu alfabeto representações gráficas e simbólicas usadas para a comunicação entre humanos ou entre homens e máquinas. A literatura apresenta alguns tipos de formalismos para a especificação de linguagens visuais. A maioria tem como base a gramática proposicional, a relacional e de grafos.

Programação Visual é definida como aquela que usa recursos visuais para a entrada de dados e para executar determinadas operações, e que permite a manipulação direta dos seus objetos pelo usuário. Como exemplo, pode-se citar uma janela composta de botões, em que cada um deles representa uma ação a ser executada quando o botão for escolhido. Se a ação desejada é a de copiar um arquivo de um periférico para outro, não é necessária a digitação do comando correspondente. Basta arrastar o ícone, que representa o arquivo, do local de origem para o de destino.

Linguagem de programação visual é definida como uma linguagem de programação mascarada por recursos visuais, como, por exemplo, BACCII, uma aplicação criada por Calloni [CAL97] que apresenta ícones representando os comandos das linguagens de programação Pascal, C e outras. Uma vez escolhidos os comandos por meio dos ícones, dentro de uma seqüência lógica, é gerado o programa na linguagem escolhida.

A visualização de software é definida como uma técnica que visa demonstrar por meio de representações gráficas o funcionamento e o comportamento das estruturas de dados e de algoritmos quanto aos aspectos dinâmicos e estáticos. Como exemplo, podem-se citar algumas ferramentas que demonstram as tabelas de símbolos utilizadas pelo compilador ou mesmo o depurador de uma linguagem de programação. Kaplan [KAP00] apresenta CUPV (Clemson University Parser Visualizer), uma extensão de CUP (Constructor of Useful Parser), ferramenta com interface gráfica que, através da interpretação, mostra os passos para a execução de uma expressão aritmética, apresentando, entre outras coisas, o conteúdo da pilha usada para a execução da expressão. Cada elemento da pilha pode ser expandido e mostrar a semântica associada a ele.

A animação de algoritmo é definida como sendo a técnica que envolve a utilização de representações visuais dinâmicas, com movimento dos dados no fluxo lógico do algoritmo. Essa característica é observada em algumas das ferramentas, descritas a seguir, que fazem a animação, por exemplo, de métodos de ordenação. Nestes casos, os dados a serem ordenados são representados por

barras coloridas de vários tamanhos e, à medida que o método é aplicado, as barras se movimentam, trocando de lugar entre si até que todo o conjunto fique ordenado.

2.4 – Considerações sobre Animação

Boroni [BOR97] estabelece uma classificação para as ferramentas de animação, dispondo-as como animação de programas, animação de algoritmo e animação de conceito.

Essa classificação pode ser comparada com as caracterizações feitas por Narayanan [NAR98] e já apresentadas. A animação de programa, para Baroni, corresponde à visualização de software descrita por Narayanan, pois a caracteriza como aquela que ilustra sobre a tela o que ocorre num programa durante sua execução, destacando as linhas que estão sendo executadas no momento, mostrando mudanças no conteúdo das variáveis e o efeito das chamadas recursivas. De certa forma é remanescente de um depurador de linguagem de programação. A animação de algoritmo é definida por ambos os autores com a mesma característica. Animação de conceito refere-se àquelas aplicações que fazem a animação de um conceito em ciência da computação; por exemplo, a demonstração de um ciclo de execução de um processador na presença de uma interrupção. Bridgeman [BRI00] apresenta PILOT (Plataform-Independent Learning Online Tool), uma ferramenta de animação de conceito.

No ensino de ciência da computação pode-se usar a potência das linguagens visuais para auxiliar os estudantes a entenderem o comportamento dos seus algoritmos e programas por meio da utilização de ferramentas de animação [MCW96].

McWirter [MCW96] sugere três situações para aplicação das ferramentas de animação em um ambiente educacional: durante uma aula teórica para demonstrar um algoritmo, estruturas de dados ou um conceito; em uma aula de laboratório onde o estudante possa interagir com a animação de algoritmos

prontos ou construídos por eles, onde porém o conjunto de animações é definido pela ferramenta; e em um ambiente onde o estudante possa construir suas próprias animações para algoritmos prontos ou construídos por ele.

A criação de ferramentas de animação teve uma expansão muito rápida à medida que os processadores foram se tornando mais poderosos, possibilitando a produção de representações visuais complexas a respeito do comportamento das estruturas de dados e de como elas atuam sobre os algoritmos [GUR96].

As ferramentas de animação, de modo geral, demonstram graficamente os dados e os processos ou métodos de um algoritmo de várias maneiras e com abordagens diferentes. Dentre as ferramentas de animação, são abordadas neste trabalho apenas aquelas que demonstram o comportamento dos dados aplicados a um determinado método ou algoritmo, representados ou não em uma estrutura de dados. Estes métodos podem ser divididos em métodos de conhecimento geral e os criados pelo usuário. São métodos de conhecimento geral aqueles métodos clássicos encontrados na literatura, como por exemplo: métodos de ordenação, de pesquisa, de busca em grafos, de manipulação de árvores binária, e outros.

A maioria das ferramentas existentes, ditas como animação de algoritmos com estruturas de dados, classifica-se nos métodos de conhecimento geral, mesmo quando se referem à animação de algoritmos que empregam estruturas de dados como a pilha ou a fila. Estas ferramentas ilustram apenas a ação das operações básicas passíveis de serem executadas sobre essas estruturas.

A animação de métodos criados pelo usuário é um processo mais sofisticado, uma vez que é necessário definir padrões para identificar, dentro da função, os elementos a serem animados e a conseqüente associação destes elementos com uma representação visual. É necessário estabelecer um conjunto básico de tipos de dados ou um conjunto de módulos, possíveis de serem agregados. Com isso, seria possível a criação de uma linguagem com recursos visuais para comunicar, de forma clara, o que o método apresenta.

2.5 – Considerações sobre Ferramentas de Animação

Os primeiros sistemas de animação, que mais se destacaram como referência para outros sistemas, como sendo os precursores, foram Balsa e TANGO. Balsa foi criado em 1984 por Brown [BRO84], que para a inclusão de recursos gráficos para entradas e saídas de dados criou BalsaII em 1988 [BRO88]. Stasko criou TANGO [STA90] e XTANGO [STA01].

Tanto Balsa quanto Tango são ambientes para a criação de animações. Vários autores usaram esses ambientes para a criação de animações de algoritmos no auxílio do aprendizado de uma linguagem ou de um conceito. Astrachan [AST98] utilizou XTANGO, como ambiente para construir a animação de um conjunto de problemas escritos em C++. O objetivo era tornar mais atraente o ensino de classes em C++. Em 1996, Pierson criou uma primeira versão do sistema JAWAA [PIE98], que é também um ambiente para criação de animação, porém voltado para a web.

Muitas ferramentas para a web, que fazem a animação de algoritmos, têm sido desenvolvidas sob a influência das facilidades de comunicação e da diversidade de recursos existentes. As ferramentas analisadas, de modo geral, são animações de métodos de conhecimento geral. Algumas delas demonstram as operações sobre as estruturas pilha, fila e listas lineares. Esta demonstração é feita utilizando-se como representação visual um arranjo ou uma lista ligada. Algorithma98 [CON99] e DSV [JAR98] são exemplos de ferramentas com essas características.

Entre as ferramentas que fazem a animação de estruturas de dados, pode-se citar DSE [ROB91] criada por Robson em 1987 e Jeliot [HAA97].

Turner, em 2001, apresentou a ferramenta Javiva [TUR01], que aborda a preocupação com a abstração envolvida no processo de aprendizado de linguagens de programação e estruturas de dados. O objetivo da ferramenta é o de auxiliar os estudantes no aprendizado de abstração através da especificação. Para isso usa o conceito de pré e pós-condições para especificar o comportamento de uma função. Considera que ensinar abstração por

especificação é difícil, uma vez que os programas usados para ensinar, por se tratar de séries iniciais, são programas simples e as especificações são úteis em programas maiores que sofrem manutenções por um longo período de tempo.

A seguir são apresentadas algumas considerações sobre as ferramentas BALSA, TANGO, JAWAA, Algorithma98, IDSV, DSE e Jeliot. Outras ferramentas que fazem a animação e/ou demonstração de elementos de linguagens de programação ou conceitos de ciência da computação, bem como ferramentas que possuem interfaces visuais para a criação e construção de aplicações, estão descritas no Anexo I.

Balsa – Brown Algorithm Simulator and Animator

Para a animação de um algoritmo são necessárias três fases. Na primeira fase, há a separação do programa em 3 componentes: o algoritmo propriamente dito, geradores de entrada que fornecem os dados a serem manipulados pelo algoritmo e as visões que deseja demonstrar em relação às ações existentes no algoritmo.

A segunda fase refere-se à implementação do algoritmo cujos eventos interessantes são marcados no código. Os eventos selecionados são transformados em chamadas de funções cujos parâmetros são o nome do evento e os argumentos associados a ele.

A terceira fase refere-se à identificação das visões e dos geradores de entradas, em que são atribuídos nomes textuais para cada algoritmo, gerador de entrada e visão. Esses nomes são apresentados ao usuário em tempo de execução em vários menus.

O usuário está restrito ao uso das visões de geradores de entrada contidos na biblioteca da ferramenta Balsa. Assim sendo, na escolha de eventos interessantes a serem marcados no algoritmo, ele está restrito à biblioteca existente na ferramenta.

Desta forma, o usuário escolhe o algoritmo a ser executado, as visões disponíveis para esse algoritmo e os geradores de entrada associados ao algoritmo.

TANGO - Transition based ANimation Generati On

TANGO [STA90] é um ambiente de desenvolvimento que habilita o usuário a criar suas próprias animações. Esse sistema faz a leitura de um arquivo escrito em ASCII ou "stdin[1]" e demonstra a animação correspondente.

XTANGO [STA01] - X-windows Transitions-based Animation Generation, que é uma versão do TANGO, teve como sucessor Samba e uma versão para Java, chamada Jsamba. Polka [STA94], também derivado do TANGO, possui versão para Windows e Unix enquanto XTANGO e Samba são para Unix [HAR94].

XTANGO é apresentado como um pacote para a animação de algoritmos de propósito geral, fornecendo resultados coloridos e em tempo real. Consiste em incluir nas funções escritas em C, diretivas que ativam rotinas de animação implementadas num arquivo separado pertencente ao pacote. Podem-se criar e manipular elementos gráficos, tais como círculos, quadrados e linhas. Ações sobre esses objetos incluem movimento, mudança de cor, tamanho, preenchimento.

JAWAA – Java and Web-Based Algorithm Animation

Outra ferramenta que habilita o usuário a criar suas próprias animações é JAWAA [PIE98]. Trata-se de um ambiente para a criação de animação de estruturas de dados para a web. Escrito em Java, permite ao usuário criar animações e então exibi-las utilizando um "browser" que suporte Java. Através de uma linha de comando escrito em JAWAA, num formato simples e um comando por linha, é possível criar animações. Existem dois tipos de comandos: os que manipulam

[1] stdin: Standard input

objetos específicos e os que especificam ações sobre um objeto. Os objetos são de dois tipos: os primitivos, como um círculo, e os inteligentes, como uma estrutura de dados que possui comandos específicos para executar operações sobre ela. Para ser usada, é necessário que o estudante tenha um conhecimento prévio sobre Java, “scripts” e programação para “web”.

Algorithma98

Algorithma98 é um pacote de software criado para auxiliar estudantes de cursos introdutórios de computação no aprendizado de métodos e estruturas de dados. Para o aprendizado de Estruturas de Dados, por exemplo, o estudante pode escolher a estrutura de fila e a partir dela escolher uma das operações oferecidas para ser executada sobre a estrutura. Neste caso, o ambiente mostra, em uma janela, o desenho de uma fila e o algoritmo que representa a operação escolhida. Para o aprendizado de um método de ordenação, por exemplo, é mostrado um conjunto de números inteiros em uma representação contígua de memória e o algoritmo do método escolhido. Neste caso, a medida que o algoritmo é interpretado, uma faixa colorida sobre o algoritmo indica a linha que está sendo executada, ao mesmo tempo em que a representação sofre as alterações relativas à ordenação.

Dentre as opções dos métodos de ordenação, é possível visualizar a demonstração, em paralelo, de dois métodos de ordenação. A apresentação dessa animação é feita com um quadro contendo pontos num plano distribuídos aleatoriamente os quais, depois de ordenados, ficam localizados numa mesma reta dentro do quadro. Ele é útil para o estudante verificar e confirmar o desempenho dos métodos. Não há a possibilidade de manipulação direta por parte do estudante sobre o ambiente, na escolha das informações a serem classificadas, por exemplo. Entretanto, é possível escolher uma estrutura de dados, a operação que deseja executar sobre essa estrutura e escolher informações que deverão ser nela inseridas.

IDSV - Interactive Data Structure Visualization

IDSV [JAR98], criado por Jarc, também é um sistema construído em Java para ser executado na web, que faz a animação de vários métodos prontos. É permitido ao estudante escolher entre ver a demonstração automática ou, com o auxílio do mouse, mostrar qual o local em que uma determinada informação deveria ser inserida em uma árvore de pesquisa, por exemplo. É anotado o nome do estudante e o seu desempenho é observado considerando-se os acertos e os erros, na compreensão dos métodos. O desempenho pode ser enviado ao professor pelo estudante. É útil para a verificação de conceitos, tais como métodos de ordenação, árvores de pesquisa, grafos. Os algoritmos foram escritos em Ada, mas a animação foi construída com “applets” em Java.

DSE - Data Structure Editor

DSE é uma ferramenta que foi criada em 1987 e que faz a interpretação e a demonstração de uma função criada pelo aluno. DSE é parte de um ambiente que consiste de um editor de sintaxe, interpretador, editor de estruturas de dados e um gerenciador de bibliotecas.

DSE, mais especificamente, permite que o estudante edite seus tipos de dados, que podem ser: arranjos, conjuntos e listas ligadas. A representação visual, que compreende o nome da variável escolhido para o tipo de dado e o valor que possui num determinado momento, é feita de forma textual e através de quadros com estas informações. As figuras contendo os quadros representando os tipos de dados são chamadas de ícones.

É possível o teste de fragmento de programas, especialmente em relação à criação de tipos de dados. Os dados criados podem ser armazenados na biblioteca para uso posterior. As funções interpretadas devem ser escritas em Modula-2 e a interface gráfica foi criada utilizando-se Xwindows.

Jeliot (Java-Eliot)

Jeliot é a versão mais recente de Eliot [LAH96], um ambiente de animação em Xwindows que utiliza primitivas de animação da biblioteca de POLKA [STA94]. Jeliot foi escrito em Java e segue o mesmo modelo de Eliot, porém para a web, com uma arquitetura adequada a este ambiente.

Para as animações em Jeliot, não é necessária a inclusão de diretivas de controle na função, como em Eliot. Para isso, é necessário utilizar os tipos de dados definidos na ferramenta. O interpretador existente no ambiente faz a identificação dos tipos de dados que a função possui e que são possíveis de animação. Os tipos de dados definidos para Jeliot incluem arranjos, filas e pilhas.

A representação visual utilizada para os tipos de dados possui o formato de círculos ou retângulos. O conteúdo da variável é mostrado dentro da figura com o respectivo nome ao lado.

Para a construção da função a ser animada deve-se utilizar um dialeto do Java criado para o ambiente, Ejava. Trata-se de um subconjunto de Java com algumas adaptações para facilitar o trabalho do interpretador, com algumas restrições de uso. O Anexo I apresenta, na figura I.6, um exemplo da representação visual de uma fila.

A representação visual das estruturas de dados fila e pilha, utilizadas em Jeliot, segue o modelo de representação por contigüidade física, isto é, semelhante a um arranjo.

2.6 – Comentários Finais

As ferramentas analisadas, apesar das especificidades que cada uma possui, e justamente por isso são diferentes umas das outras, podem ser divididas em dois grupos: (1) ferramentas que fazem a animação de uma função criada pelo usuário; e (2) ferramentas que fazem a animação de métodos de conhecimento geral, fornecidos pela ferramenta.

A Figura 2.1 mostra uma tabela com as ferramentas que compõem os dois grupos, destacando as características necessárias para a construção de ADTTool.

	Ferramenta	Interpreta função	Linguagem da função interpretada	Interface Gráfica	Incluir Diretivas na Função	Forma de Representar		
						Pilha	Fila	Lista Linear
1° grupo	BALSA I	Sim	Pascal	XWindows	Sim	depende do usuário	depende do usuário	depende do usuário
	XTANGO	Sim	C	XWindows	Sim	depende do usuário	depende do usuário	depende do usuário
	JAWAA	Sim	Java	Applets	Sim	contigüidade física	contigüidade física	não representa
	DSE	Sim	Pascal	XWindows	Não	não representa	não representa	não representa
	Jeliot	Sim	Java	Applets	Não	contigüidade física	contigüidade física	não representa
2° grupo	Algorithma98	Não	-	Applets	-	lista ligada	lista ligada	não representa
	IDSV	Não	-	Applets	-	não representa	não representa	não representa

Figura 2.1 – Tabela comparativa das ferramentas dos grupos 1 e 2

No primeiro grupo, destacam-se BALSA, XTANGO, JAWAA, DSE e Jeliot. As três primeiras, apesar de fazerem a animação de algoritmos criados pelo usuário, isto é, interpretam uma função, são ambientes para a criação de animações. Para utilizar estas ferramentas, é necessário associar diretivas de controle aos elementos da função em que se deseja a animação.

Para testar uma função em um desses ambientes, é necessário que o estudante aprenda os comandos de controle para animação e os associe aos elementos de sua função para produzir a animação. O estudante deve, então, decidir sobre os elementos da função que deseja ver demonstrados na tela, bem como a forma gráfica de cada um, e os momentos que geram movimentos nestes elementos. Isto significa que a forma para representar as pilhas, filas e lista linear dependem da escolha de quem está construindo a animação.

Entretanto, o objetivo da matéria em questão não é oferecer recursos para que o estudante crie suas animações e sim aprenda, através da visualização dos elementos de sua função, o uso das estruturas de dados como tipo de dado abstrato.

DSE e Jeliot são ferramentas que fazem a animação de uma função criada pelo usuário; possuem um interpretador que identifica os tipos de dados a serem animados. Entretanto, as representações visuais utilizadas tanto por DSE quanto por Jeliot não atendem à necessidade discutida neste trabalho sobre não utilizar recursos que induzam os estudantes à forma de armazenamento destas estruturas.

DSE não oferece recursos específicos para a edição das estruturas de dados: lista linear, pilha e fila. Entretanto, através de quadros e tabelas, é possível simular uma pilha, por exemplo.

Jeliot prevê o uso das estruturas de dados: filas e pilhas. Entretanto, a representação visual é feita através de retângulos sobrepostos semelhantes à representação de um arranjo (contigüidade física).

O segundo grupo, formado pelas ferramentas de animação de métodos de conhecimento geral, têm como característica principal mostrar o funcionamento desses métodos através de uma representação gráfica, dinâmica e instrutiva. Além das citadas, Algorithma98 e IDSV, existem várias outras semelhantes, descritas no Anexo I. Várias delas apresentam animações com as estruturas de dados (pilhas e filas), com a apresentação da ação das operações de inserção e eliminação de informações sobre estas estruturas. Entretanto, o estudante não tem dúvidas sobre a ação das operações de inserção e eliminação de elementos no topo de uma pilha, por exemplo. A maior dificuldade está em empregar as estruturas de dados na solução de problemas.

Para atender a necessidade de executar uma função, criada pelo estudante, que emprega no seu desenvolvimento uma estrutura de dado como um tipo de dado abstrato, a ferramenta deveria interpretar e demonstrar uma função escrita em C e que, principalmente, mantivesse a abordagem abstrata vinculada ao tipo de dado.

Nenhuma das ferramentas apresentadas possui um formalismo teórico de construção de um tipo de dado abstrato e de uma representação visual para uso, ainda como componente abstrato, na solução de problemas. Nenhuma delas

demonstra uma função com o objetivo de manter o nível de abstração empregado na criação do tipo de dado.

A ferramenta ADTTool, apresentada no Capítulo V, foi desenvolvida com a finalidade de preencher este vazio.

ADTTool trata apenas da abstração envolvida com a aplicação das estruturas básicas de dados (pilhas, filas e listas lineares), em problemas que necessitem delas; não há necessidade de implementação destas estruturas e suas operações. ADTTool foi criada para o estudante testar funções isoladas, que foram construídas usando as estruturas de dados e suas operações, isto é, sem ter ainda decidido onde armazená-las. Apesar deste protótipo permitir apenas funções que não contenham chamadas de outras funções, e possuir um conjunto limitado de tipos de dados, ele permite que o estudante crie suas próprias funções e as teste no ambiente, gerando uma animação da execução das mesmas.

Com relação às estruturas de dados suportadas, muitas dessas ferramentas, ao se referirem às estruturas de dados, não apresentam a lista linear como uma das estruturas de dados e sim a lista ligada. A lista ligada é uma das formas de representação interna de uma lista linear. A partir do momento em que o estudante vê sua lista linear representada numa lista ligada, deixa de pensar de forma abstrata sobre a estrutura e passa a pensar na forma de implementação, isto é, detalhes de implementação que deveriam estar escondidos foram revelados. O estudante tende, então, a pensar em sua lista com elementos de ligação e endereços, que são detalhes de implementação de uma lista ligada.

A lista ligada e os arranjos são formas de representação interna e para que as estruturas de dados não fossem associadas a qualquer destas formas de representação, ADTTool possui uma representação visual que não se prende a recurso algum de representação interna.

ADTTool é uma ferramenta que possui uma *linguagem visual* própria, desenvolvida especialmente para demonstrar o funcionamento de uma função, elaborada pelo estudante utilizando a linguagem de programação C. Para a composição da linguagem visual foram criadas *representações visuais* especiais para cada uma das estruturas básicas de dados, que foram denominadas de

representações visuais abstratas. A interface da ferramenta por ser composta de botões, para a escolha de ações a serem executadas, atende à característica de uma programação visual. A ADTTool realiza a animação do programa ao mostrar o movimento dos dados de uma estrutura para outra e à modificação dos valores associados às variáveis simples.

CAPÍTULO III

CONSIDERAÇÕES SOBRE INTERFACES ORIENTADAS AO USUÁRIO

Introdução

Neste capítulo são apresentados alguns aspectos envolvidos na construção de uma interface orientada ao usuário. Alguns itens destacam sua evolução e as dificuldades atuais em relação às ferramentas existentes para a sua construção. Aborda também as principais características ligadas à medida de usabilidade de uma interface e apresenta alguns exemplos de ferramentas para a criação de interfaces gráficas.

3.1 – Sobre a Usabilidade da Interface Orientada ao Usuário

Um sistema, antes de tudo, deve possuir um alto grau de aceitação, deve ser útil para atender uma determinada funcionalidade, e a sua usabilidade deve responder à questão: Qual a melhor forma que um sistema pode atender ao usuário?

A melhor forma para atender as expectativas do usuário em relação a uma interface envolve, basicamente, duas partes: a interna e a externa, ou seja, a parte que o usuário não vê e a que ele vê. A Figura 3.1 mostra a interface separando as partes interna e externa da comunicação.

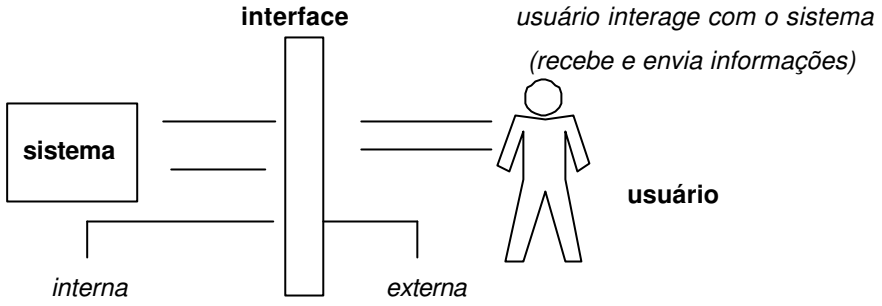


Figura 3.1 – Interface separando a parte interna da externa

A parte interna diz respeito ao desenvolvimento do sistema, que basicamente deve atender aos requisitos mínimos de confiabilidade e à funcionalidade desejada, além de traduzir as informações oferecidas pelo e ao usuário, através da interface.

A parte externa diz respeito à interface que fará a comunicação do sistema com o usuário. Refere-se à visão que o usuário tem do sistema e é nela que estará depositada toda a potencialidade de usabilidade que o sistema pode oferecer.

Apesar de estar dividida em partes, a interface não pode ser considerada simplesmente como um elemento que deverá ser pensado na fase final do processo de desenvolvimento de um software. A forma de comunicação das entradas e saídas do sistema são elementos inerentes ao sistema e a forma como esses processos ocorrerão não pode ser deixada para uma decisão ao final do processo.

Dizer que a interface é voltada ao usuário significa colocá-lo como foco principal do sistema. A preocupação com a construção da interface quanto à definição de seus elementos, funcionamento, aparência, formas de interação, deve levar em consideração o seu público alvo, seu nível de conhecimento do sistema e das ferramentas utilizadas, bem como fatores humanos envolvidos na sua comunicação com o sistema.

Nielsen [NIE93] destaca alguns princípios básicos para a usabilidade a serem observados para a criação das interfaces. Os diálogos devem ser simples e naturais, e as informações deveriam aparecer numa ordem lógica. É importante falar a linguagem do usuário, usar um vocabulário compatível com a idade, grau de conhecimento do sistema, ao invés de utilizar termos orientados a ele. Deve-se evitar a sobrecarga de informações inter-relacionadas, que obriguem o usuário a lembrar informações de outras partes do sistema, obrigando-o a voltar a passos anteriores. É necessário que o mesmo comando produza sempre o mesmo resultado ou que um determinado item da interface possua sempre a mesma localização. Mensagens curtas e apropriadas sobre o que está acontecendo ajudam a aumentar a segurança do usuário. Em casos de entradas de dados, é necessário informar as condições e formatos para evitar erros. Teclas aceleradoras, para determinadas funções, devem ser empregadas para facilitar a tarefa de usuários mais experientes. As mensagens de erros devem ser claras. Devem-se prevenir erros. Ajudas (*Help*) e documentações são elementos fundamentais para as aplicações.

Mesmo ao se considerarem todas as regras e padrões existentes para a construção de uma interface, a fim de atender os requisitos de usabilidade, existem determinados elementos da aparência da interface que não possuem regras. Elas fazem parte da criatividade de cada projetista.

O projetista pode ser influenciado pela observação de outros sistemas, pelo meio social e cultural em que vive, por dons próprios, conhecimentos adquiridos e experiências vividas em outras áreas. Essas influências envolvem a sua criatividade e afetam o seu projeto.

A criatividade pode ser observada mesmo em programas produzidos por estudantes de uma mesma classe cuja forma de apresentação dos resultados é diferente. Alguns não possuem a percepção de uma apresentação clara, mesmo para a produção de resultados na forma de texto.

O uso de recursos simples como tabulação, títulos e palavras sublinhadas não são elementos naturais percebidos por todos. Considerando um conjunto

maior de possibilidades como cores e formas geométricas, a combinação desses elementos pode não produzir um resultado atraente.

A criatividade, como apoio ao desenvolvimento de interfaces, pode ser trabalhada através de um conjunto de atividades. Shneiderman [SHN00] apresenta *genex* (*generator of excellence*), um esquema de trabalho composto de quatro fases e oito atividades, voltado para a produção de interfaces criativas.

A interface como elemento de comunicação do sistema com o usuário deve respeitar suas características tanto em relação aos fatores humanos envolvidos quanto à forma de transmitir ou receber a informação entre ambos.

3.2 – Sobre o Usuário e os Estilos de Interface

Uma das classificações feitas sobre o tipo de usuário das interfaces diz respeito à sua experiência com o sistema, com computadores em geral e com o domínio da tarefa a ser executada. A variação do nível de conhecimento dos três elementos para um usuário é representada por Nielsen [NIE93] em um gráfico tridimensional, uma vez que um usuário não se insere em apenas uma das classificações anteriores.

O nível de conhecimento do usuário em relação ao sistema pode ser mínimo. Porém, este deve possuir bons conhecimentos do domínio da aplicação. Assim, o nível de conhecimento de um usuário está ligado ao tipo de sistema usado e ao estilo da interface nele empregada.

A manipulação direta, como um dos estilos de interface, refere-se aos sistemas em que é permitido ao usuário manipular os objetos mostrados na tela do computador, simplesmente através da escolha de um deles, usando algum tipo de dispositivo de apontamento. Alguns dos recursos associados a esse estilo são o *mouse*, janelas, menus suspensos. Com a manipulação direta, toda a programação é feita através de uma representação gráfica, de maneira que os elementos expressem uma idéia o mais próxima possível da forma de pensamento do usuário, ao invés de usar um recurso computacional, muitas vezes abstrato para ele [HUT86]. Estas representações gráficas que refletem um pensamento ou

uma idéia são apresentadas nas interfaces, por exemplo, através dos ícones e botões.

A manipulação direta permite aprender rapidamente as funcionalidades básicas através de uma demonstração feita por outro usuário experiente ou através de tutores. A manipulação direta também permite visualizar imediatamente o resultado de suas ações [SHN98]. Além disso, uma interface que oferece ao usuário a manipulação direta de seus elementos, ao exibi-los induz o usuário a associá-los à tarefa que deseja executar, ao mesmo tempo em que o encoraja a acioná-los. Essa possibilidade auxilia na redução da ansiedade dos usuários com pouca ou nenhuma experiência com os sistemas que operam.

Outros estilos de interface referem-se à seleção de menus, formulários para preenchimento, linguagens de comando e linguagens naturais [SHN98].

Os menus de seleção oferecem ao usuário a possibilidade de executar determinada tarefa através da escolha de elementos em uma lista apresentada. O resultado de cada escolha pode ser visualizado imediatamente na maioria das vezes. Com os elementos colocados em menus, o usuário associa sua necessidade aos elementos listados, aumentando sua segurança na escolha.

Os formulários com campos para preenchimento oferecem ao usuário rótulos indicando que informação lhe deve ser fornecida. Entretanto, instruções sobre o formato da informação a ser inserida ou sobre os valores válidos nem sempre estão claros para o usuário, sendo possível apenas para aqueles mais experientes. Com isso, algumas aplicações necessitam de treinamento ou são indicadas apenas para usuários mais experientes.

As linguagens de comandos são indicadas para usuários com experiência intermediária de conhecimento do sistema. Devem aprender a sintaxe da linguagem ou recorrer aos tutores oferecidos.

As linguagens naturais representam a comunicação futura. Pesquisadores trabalham para encontrar meios de efetuar esta forma de comunicação.

Uma interface com recursos gráficos e com vários estilos de comunicação, de forma a melhor atender às necessidades e anseios do usuário, deve ser construída com ferramentas que atendam a estas características.

3.3 – Ferramentas para Construção de Interfaces Gráficas

As ferramentas para a construção de interfaces causaram um grande impacto na prática de desenvolvimento de software, uma vez que a necessidade dos recursos gráficos para as interfaces obrigou os sistemas mais antigos a passarem por um processo de reformulação, incorporando recursos gráficos para as interfaces.

Os gerenciadores de janelas e o estilo gráfico das interfaces são resultados de pesquisas efetuados nos anos 70, pela Xerox Palo Alto Research Center (PARC), no MIT e no Instituto de Pesquisa de Stanford [MYE00].

A partir de então, outros institutos de pesquisa investiram muitos recursos para a produção de ferramentas utilizadas na construção de interfaces, sendo a primeira delas a linguagem orientada a eventos chamada *HyperTalk* e o *Visual Basic*. A seguir vieram as gerações atuais como *Object Linking and Embedding* (OLE) da *Microsoft* e Java *Beans.*

Um dos programas de maior sucesso de todos os tempos é a planilha e o motivo do sucesso refere-se ao fato das facilidades oferecidas ao usuário de poder programar fórmulas e macros, sem o conhecimento de uma linguagem de programação [MYE00]. Esse também tem sido o sucesso da *internet* cuja construção de uma página, sem elementos sofisticados, não exige muitos conhecimentos além dos necessários para um editor de texto.

O uso de determinadas ferramentas para a construção de interfaces ainda é muito difícil de aprender quanto o é de programar. Um estudo feito por Myer [MYE00] sobre estas ferramentas utilizadas na construção de interfaces mostra que, para cada melhoria ou sofisticação incorporada a uma interface já em uso, considerando-se a visão do usuário, é necessário que ele adquira apenas alguns poucos conhecimentos para a compreensão destas novas modificações efetuadas no sistema. Muitas vezes essas melhorias tornam ainda mais fácil o uso do sistema. Considerando-se a visão do projetista, para efetuar uma melhoria ou sofisticação através da inclusão de novos recursos no sistema, é necessário que ele adquira novos conceitos, nem sempre simples. Em algumas etapas da

construção de um sistema ou incorporação de melhorias, é necessário que o projetista pare para aprender novos conceitos e técnicas para então avançar no uso da ferramenta.

O gráfico apresentado na Figura 3.2, mostra uma versão resumida do apresentado por Myer [MYE00]. O objetivo do gráfico é mostrar, sob o ponto de vista do projetista, o grau de dificuldade do uso de algumas ferramentas para a criação da interface de uma aplicação com diferentes níveis de sofisticação que ela pode ter. Os pontos de inflexão representam os momentos onde é necessária a aquisição, por parte do projetista, de algum conhecimento para aumentar o grau de sofisticação da aplicação.

No gráfico, a representação de uma ferramenta que utiliza a programação em C possui grau de dificuldade inicial maior do que a programação que utiliza *Visual Basic*.

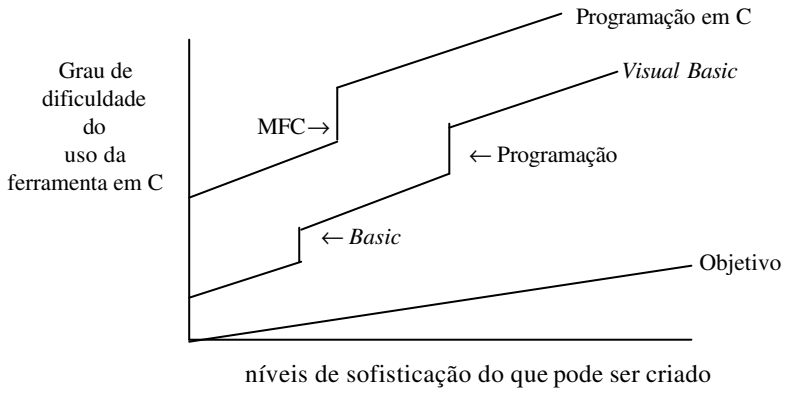


Figura 3.2 – Relação entre dificuldade de projeto e sofisticação da interface (fonte: [MYE00])

A construção de uma interface, utilizando-se uma ferramenta baseada na programação C, encontra a primeira barreira com a necessidade de aprender *Microsoft Foundation Class* (MFC), para a inclusão de representações gráficas nas interfaces.

Com o *Visual Basic* há dois momentos de parada para a obtenção de conhecimento. O primeiro refere-se ao conhecimento inicial da linguagem, usando o ambiente com os recursos “clique e crie” a partir de ícones, menus e caixas de

diálogos. O segundo refere-se ao conhecimento da programação para a integração da aplicação com as representações gráficas que compõem a interface.

O grau de dificuldade encontrado para a criação de interfaces gráficas com o uso de recursos visuais, como em MFC, tem propiciado o aparecimento de várias ferramentas que oferecem um ambiente mais simples de se utilizar na criação das interfaces. Entre as mais citadas na literatura, destacam-se: Amulet [MYE97], PUIST [LI 97] e JPT [RAS01]. A descrição destas ferramentas encontra-se no Anexo I.

3.4 – Comentários Finais

Com o advento das aplicações orientadas a janelas e botões, dificilmente uma aplicação teria boa aceitação se não fosse criada segundo esse paradigma.

A partir disso, apesar do grau de dificuldade na utilização dos elementos gráficos envolvidos com as classes do MFC, ADTTool foi construída utilizando as classes do MFC, em função da grande potencialidade de recursos que a linguagem C++ e as classes MFC oferecem.

A interface foi criada com elementos padrão para janelas e botões, que não exigem um conhecimento adicional do usuário para a sua manipulação; embora esteja voltada para um grupo específico de usuários que são os estudantes do curso de estruturas de dados.

Toda a linguagem empregada em mensagens de erros e apresentação dos resultados segue o mesmo padrão utilizados durante as aulas teóricas.

CAPÍTULO IV

FORMALIZAÇÃO DAS ESTRUTURAS DE DADOS

Introdução

Neste capítulo é feita uma descrição sobre a importância de se aprender a empregar a abstração no desenvolvimento de projetos e da dificuldade dos estudantes em compreender e usar uma estrutura de dados como um Tipo de Dado Abstrato (TDA). É proposta uma redefinição dos componentes de um TDA, levando-se em consideração apenas aspectos abstratos, dentro do que se pretende atingir com o ensino das estruturas de dado, e a inclusão de um novo componente, também de característica abstrata, que deverá fazer a ligação entre a representação teórica do TDA e a sua representação visual.

4.1. Definição de um Tipo de Dado Abstrato (TDA)

Segundo Sengupta [Sen94] um Tipo de Dado Abstrato (TDA) é uma coleção de dados e um conjunto de operações que são utilizadas para definir e manipular os dados.

A partir da definição, pode-se dizer que um tipo de dado agrega dois componentes:

- campo de definição dos dados;
- operações definidas para a manipulação desses dados.

A abstração associada ao tipo de dado significa que o uso de um tipo de dado abstrato não depende de recursos de uma linguagem de programação, isto é, o uso das operações definidas para o TDA deve ser feito, independentemente do meio e forma que serão escolhidos para o armazenamento dos dados.

A escolha da forma de armazenamento dependerá da aplicação que fará uso das estruturas de dados e das operações que os manipulam. Apesar disto, as operações não sofrerão modificações, pois são tipos de dados abstratos definidos segundo esse conceito.

A Figura 4.1 mostra um esquema de como separar o problema principal das operações, resultando uma camada de abstração e uma camada de funcionalidade. A camada de abstração deverá conter as operações definidas para o tipo de dado abstrato e deve ficar escondida da aplicação principal, que é a camada da funcionalidade do problema. A camada da funcionalidade é onde se encontra a função principal, que faz uso do tipo de dado abstrato e de suas operações e que precisa ter sua funcionalidade testada.

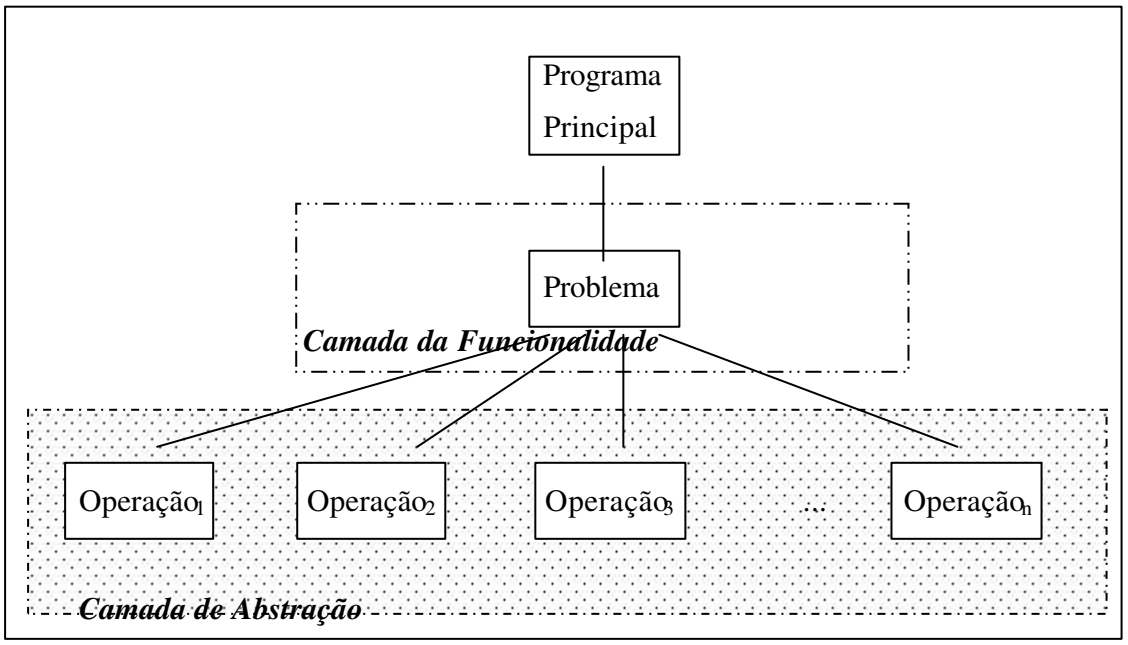


Figura 4.1 – Esquema de representação do problema

A criação de uma estrutura de dados como TDA significa, inicialmente, definir o comportamento e as restrições dos dados sobre essa estrutura. Essa definição deve ser feita através do uso de elementos formais atendendo um grau de abstração, sem utilizar recursos ligados a linguagens de programação. Quando se fala em campo de definição dos dados pretende-se referir ao comportamento e restrições dos dados. Assim sendo, esse campo de definição dos dados passa a ser chamado de ***definição abstrata dos dados***.

A caracterização das estruturas como TDA tem gerado uma grande dificuldade na compreensão da abstração e consequentemente em uma resistência ao uso das operações. Como conseqüência, os estudantes não separam o problema das operações, não separando a camada da funcionalidade do problema da camada de abstração.

A grande dificuldade encontrada ao ensinar estruturas de dados sob esse enfoque está em mostrar que é possível construir módulos usando as operações sem vinculá-las às formas de armazenamento. Normalmente, o que se encontra na literatura é a especificação de uma pilha, por exemplo, como tipo de dado abstrato. Mas a demonstração de sua forma é feita através de um arranjo unidimensional. Essa representação visual se caracteriza num elemento concreto da linguagem de programação e induz o estudante à utilização dos elementos da pilha como uma posição do arranjo, P[i], em lugar de usar as operações definidas para a pilha. A pilha tratada dessa forma deixa de ser um tipo de dado abstrato. Ser abstrato significa que serão definidas ações sobre essa pilha, ações estas que provocarão mudanças no seu conteúdo e que deverão valer para qualquer tipo de armazenamento. Assim sendo, essas ações passam a ser chamadas de ***ações abstratas***.

Para a demonstração de um tipo de dado abstrato, através da manipulação das ações abstratas definidas para ele, de forma a melhor compreender e enxergar o resultado da ação de uma operação sobre o tipo, foi necessária a criação de uma ***representação visual abstrata*** associada à definição de TDA. A representação abstrata passa a ser o terceiro componente de um tipo de dado abstrato. É importante agregar a palavra abstrata à representação uma vez que se

pretende demonstrar a ação das operações através de um modelo com elementos que não dependem de qualquer aspecto da linguagem de programação, garantindo que o tipo de dado continue abstrato. Assim sendo, os modelos criados para as estruturas de dados básicas, listas lineares, filas e pilhas, apesar de bastante simples, foram escolhidos de forma a não induzir, através da sua representação, a escolha por uma representação interna , tais como, a contiguidade física (como arranjos, tabelas ou arquivos) ou ligada (tanto na forma dinâmica quanto na estática) escondendo, assim, detalhes sobre o armazenamento dos elementos.

Com a inclusão da representação abstrata, a definição de um tipo de dado abstrato seria:

Definição: um tipo de dado abstrato é um tipo de dado que agrega *três* componentes:

- definição abstrata dos dados;
- ações abstratas;
- uma representação abstrata.

4.2 – Definição Formal das Estruturas de Dados

A especificação formal de um tipo de dado abstrato pode ser feita através de qualquer linguagem de descrição. Alguns autores optaram pela adoção de uma linguagem própria, como é o caso de Horowitz que criou a SPARKS [HOR87]. Outros adotaram a sintaxe de alguma linguagem de programação, tomando o cuidado de não vincular a definição com as características específicas da linguagem.

Para a especificação formal das estruturas de dados básicas, como tipos de dados abstratos, apresentada a seguir, foi usada a combinação de duas formas de especificação: a descritiva, para descrever os aspectos relativos à ação, os elementos que compõem as entradas e os resultados esperados; e a algébrica,

para a descrição da sintaxe das ações e dos axiomas ligados ao tipo de dado abstrato que se está definindo.

Assim, as principais estruturas de dados trabalhadas com os estudantes: lista linear, fila e pilha, seguem com a seguinte característica:

LISTA:

Definição abstrata dos dados: lista linear é um conjunto de $n \geq 0$ elementos de qualquer tipo, x_1, x_2, x_3, ... , x_n cujas propriedades estruturais envolvem somente a posição relativa linear do nó. Isto é, para $n>0$, x_1 é o 1º elemento, x_2 é o 2º elemento, e para um certo k $(1 < k < n)$ x_k é o k-ésimo elemento, x_k é precedido por x_{k-1} e seguido por x_{k+1} e, x_n é o último elemento. Para n=0 a lista linear está vazia.

Ações abstratas: são executadas sempre em relação à posição relativa linear.

Iniciar:

ação : tornar a lista vazia, pronta para receber informações;

entrada : a lista que deverá ser transformada em vazia;

resultado : a lista vazia, pronta para receber as informações e comprimento da lista igual a zero;

sintaxe : *init* (lista) $\rightarrow$ lista vazia.

Inserir:

ação : inserir uma informação na i-ésima posição da lista;

entrada : a lista que receberá a informação, a informação a ser inserida e a posição que receberá a informação;

resultado : a lista alterada e comprimento adicionado de uma unidade;

sintaxe : *inserir* (lista, valor , posição) → lista com o elemento inserido.

Eliminar:

ação : eliminar uma informação da i-ésima posição da lista;

entrada : a lista que sofrerá a eliminação e a posição de onde deverá ser eliminada a informação;

resultado : a lista alterada, comprimento diminuído de uma unidade, a informação retirada;

sintaxe : *eliminar* (lista, posição) → elemento eliminado e lista sem o elemento.

Consultar:

ação : consultar uma informação contida na i-ésima posição da lista;

entrada : a lista para a consulta e a posição de onde deverá ser lida a informação;

resultado : a informação consultada (lista e comprimento inalterados);

sintaxe : *consultar*(lista, posição) → cópia do elemento da i-ésima posição.

Alterar:

ação : alterar o conteúdo da i-ésima posição da lista com uma nova informação;

entrada : a lista, a posição a ter seu conteúdo modificado e nova informação;

resultado : a lista modificada;

sintaxe : *alterar* (lista, valor, posição) $\rightarrow$ lista modificada.

Comprimento:

ação : devolver a quantidade de elementos na lista;

entrada : a lista a ser verificado o comprimento;

resultado : a quantidade de elementos da lista;

sintaxe : *compr*(lista) $\rightarrow$ comprimento da lista.

Vazia:

ação : verificar se a lista está vazia ou não;

entrada : a lista a ser analisada;

resultado : verdadeiro (vazia) ou falso (não vazia);

sintaxe : *vazia* (lista) $\rightarrow$ verdadeiro ou falso.

As operações definidas para a lista linear se dividem em ações de construção e de inspeção:

ações de construção	ações de inspeção
init (lista)	consultar(lista, posição)
inserir (lista, valor, posição)	compr (lista)
eliminar (lista, posição)	vazia (lista)
alterar (lista, valor, posição)	

Considere as definições:

$L \in$ lista,

$pos, max \in$ inteiros; onde max : número máximo de elementos

$valor \in$ tipo_do_elemento(*);

$b \in \{V, F\}$

REGRAS DE VALIDAÇÃO:

01. Erro de Sintaxe – Posição Inválida

a) inserir (L, $valor$, pos) $\rightarrow$ Erro, se $pos \leq 0$ ou $pos >$ compr (L) + 1;
b) consultar (L, pos) $\rightarrow$ Erro, se $pos \leq 0$ ou $pos >$ compr (L);
c) eliminar (L, pos) $\rightarrow$ Erro, se $pos \leq 0$ ou $pos >$ compr (L);
d) alterar (L, $valor$, pos) $\rightarrow$ Erro, se $pos \leq 0$ ou $pos >$ compr (L);

02. Erro – Overflow ou Underflow

a) inserir (L, $valor$, pos) $\rightarrow$ Erro de "overflow", se compr (L) = max
b) eliminar (init (L), pos) $\rightarrow$ Erro de "underflow", $\forall pos$;

(*) tipo_do_elemento é um tipo definido pelo usuário de acordo com a aplicação.

AXIOMAS:

01. consultar (init (*L*), *pos*) → erro;
02. consultar (inserir (*L*, *valor*, *pos*), *pos*) → *valor*;
03. consultar (eliminar (*L*, *pos*) , *pos*) → consultar (*L*, *pos* +1);
04. consultar (alterar (*L*, *valor*, *pos*), *pos*) → *valor*;
05. compr (init (*L*)) → 0;
06. compr (inserir (*L*, *valor*, *pos*)) → compr (*L*) + 1;
07. compr (eliminar (*L, pos*)) → compr (*L*) –1;
08. compr (alterar (*L, valor, pos*)) → compr (*L*);
09. vazia (init (*L*)) → V;
10. vazia (inserir (*L*, *valor*, *pos*))→ F;
11. vazia (eliminar (*L* , *pos*))→ V se compr (*L*) = 1 e
    F se compr (*L*) > 1;
12. vazia (alterar (*L*, *valor*, *pos*)) → F;

Representação Abstrata: a lista linear deve ser representada destacando-se apenas as posições relativas lineares que caracterizam a sua definição e o nome dado a ela. Assim sendo, não se está vinculando a lista linear a qualquer forma de armazenamento interno. Veja exemplo na Figura 4.2.

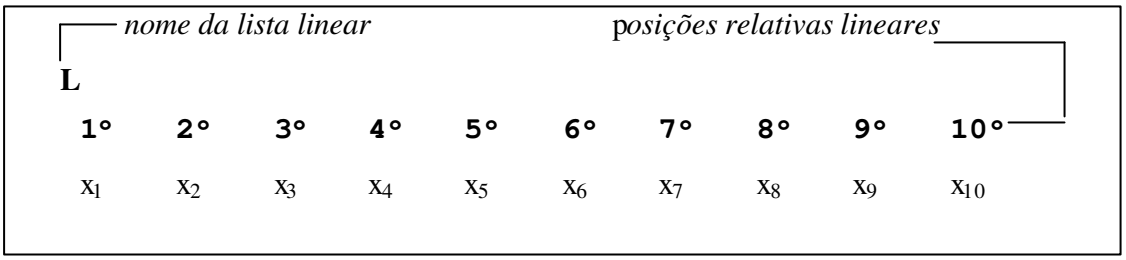


Figura 4.2: Representação Abstrata da Lista Linear

Cada elemento x_i representa um elemento da lista e ocupa a i-ésima posição na lista linear.

Deve ficar claro na execução das ações que os resultados são produzidos nos dados contidos na lista linear, independentemente do meio de armazenamento. Assim sendo, no exemplo abaixo, após cada aplicação de uma ação, apresenta-se a representação abstrata da lista linear resultante dessa ação.

O exemplo da Figura 4.3 parte de uma representação inicial de uma lista linear, com cinco elementos quaisquer, representados por letras gregas, e uma seqüência de ações. Após cada uma delas, é mostrado o resultado obtido com aplicação da ação.

Considere as seguintes declarações:

$L \in$ lista;

$\alpha, \beta, \phi, \gamma, \varphi, \lambda, \xi$ (constantes) $\in$ tipo_do_elemento;

$b \in \{ V, F \}$;

$x \in$ tipo_do_elemento;

lista inicial:

L

1º	**2º**	**3º**	**4º**	**5º**
α	β	ϕ	γ	φ

(I) aplicar a ação: ***inserir (L,*** λ***, 3)*****;**

L

1º	**2º**	**3º**	**4º**	**5º**	**6º**
α	β	λ	ϕ	γ	φ

(II) aplicar a ação: ***inserir (L,*** ξ***, compr(L)+1);***

L

1º	**2º**	**3º**	**4º**	**5º**	**6º**	**7º**
α	β	λ	ϕ	γ	φ	ξ

(III) aplicar a ação **: *x = eliminar(L, 1);***

L

1º	**2º**	**3º**	**4º**	**5º**	**6º**
β	λ	ϕ	γ	φ	ξ

x ← α

(IV) aplicar a ação **: *b = vazia (L);***

L

1º	**2º**	**3º**	**4º**	**5º**	**6º**
β	λ	ϕ	γ	φ	ξ

b ← falso

(V) aplicar a ação **: *x = consultar (L, 5);***

L

1º	**2º**	**3º**	**4º**	**5º**	**6º**
β	λ	ϕ	γ	φ	ξ

x ← φ

Figura 4.3: Representação da lista *L* em uma seqüência de operações

PILHA:

Definição abstrata dos dados: a pilha é uma lista linear com restrições nas ações de inserção e eliminação que devem ocorrer sempre em uma extremidade da estrutura, chamada *topo* da pilha.

Ações abstratas: são executadas sempre em relação ao topo da estrutura.

Iniciar:

ação : tornar a pilha vazia, pronta para receber informações;
entrada : a pilha que deverá ser transformada em vazia;
resultado : a pilha vazia;
sintaxe : *init* (pilha) $\rightarrow$ pilha vazia.

Desempilhar:

ação : eliminar o elemento de topo, ajustando o controlador de topo;
entrada : a pilha que sofrerá a eliminação;
resultado : o elemento do topo e a pilha modificada, isto é, sem o elemento do topo;
sintaxe : *pop* (pilha) $\rightarrow$ elemento do topo e pilha sem o elemento do topo.

Empilhar:

ação : empilhar um elemento no topo da pilha, ajustando o controlador de topo;
entrada : a pilha e o elemento a ser empilhado;
resultado : a pilha modificada, isto é, com o elemento novo no topo da pilha;
sintaxe : *push*(pilha, valor) $\rightarrow$ pilha acrescido do novo elemento no topo.

Vazia:

ação : verificar se a pilha está vazia ou não;
entrada : a pilha a ser analisada;
resultado : resposta se vazia (verdadeiro) ou não (falso);
sintaxe : *vazia*(pilha) $\rightarrow$ verdadeiro ou falso.

Consultar topo:

ação : retornar o elemento do topo da pilha sem eliminá-lo;
entrada : a pilha a ter seu topo consultado;
resultado : o elemento do topo da pilha, sem eliminá-lo. A pilha fica inalterada;
sintaxe : *topo* (pilha) $\rightarrow$ cópia do elemento do topo da pilha.

As operações definidas para a pilha se dividem em ações de construção e de inspeção:

ações de construção	ações de inspeção
init (pilha)	topo(pilha)
push (pilha, valor)	vazia (pilha)
pop (pilha)	

Considere as seguintes definições:

$P \in$ pilha;
max $\in$ inteiros; *max*: número máximo de elementos
valor $\in$ tipo_do_elemento
$b \in \{ V, F\}$

REGRAS DE VALIDAÇÃO:

01. Erro – Overflow e Underflow
    a) push (*P*, *valor*) → Erro de “overflow”, se e somente se a pilha possui *max* elementos
    b) pop (init (*P*)) → Erro de “underflow”;
    c) topo (init (*P*)) → Erro de “underflow”;

AXIOMAS:

01. vazia (init (*P*))→ V;
02. vazia (pop (*P*))→ V se *P* possui apenas um elemento e
    F se *P* possui mais de um elemento;
03. vazia (push (*P*, *valor*))→ F;
04. topo (init (*P*))→ Erro de Underflow;
05. topo (pop (*P*))→ *valor*, elemento abaixo do topo (*P*);
06. topo (push (*P*, *valor*))→ *valor*;

Representação Abstrata: a pilha deve ser representada destacando-se apenas a característica agregada à sua natureza, de se colocar uma informação sobre a outra, o indicativo do topo da pilha e o nome dado a ela. Veja exemplo na Figura 4.4.

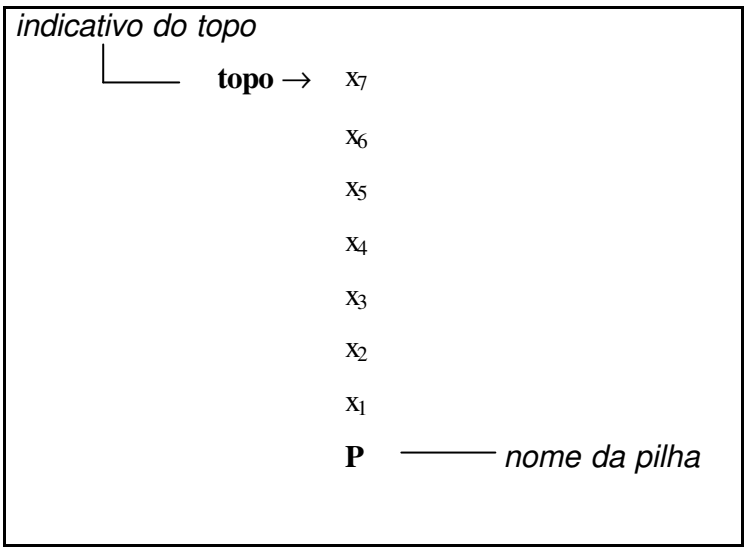


Figura 4.4: Representação Abstrata de uma Pilha

O esforço concentrado no uso de uma pilha está em compreender que as operações pertinentes a essa estrutura devem atuar sempre em relação ao topo da pilha.

A Figura 4.5 mostra um exemplo do uso das ações definidas para a pilha através de sua representação abstrata, partindo de uma configuração inicial da estrutura, contendo elementos constantes quaisquer, representados por letras gregas. Uma seqüência de operações é apresentada e após cada uma delas é demonstrado o resultado obtido sobre a pilha após sofrer a sua aplicação.

Considere as seguintes declarações:

$P \in$ pilha;

$\alpha, \beta, \phi, \gamma, \varphi, \lambda$ (constantes) $\in$ tipo_do_elemento;

$b \in \{ V, F \}$;

$x \in$ tipo_do_elemento;

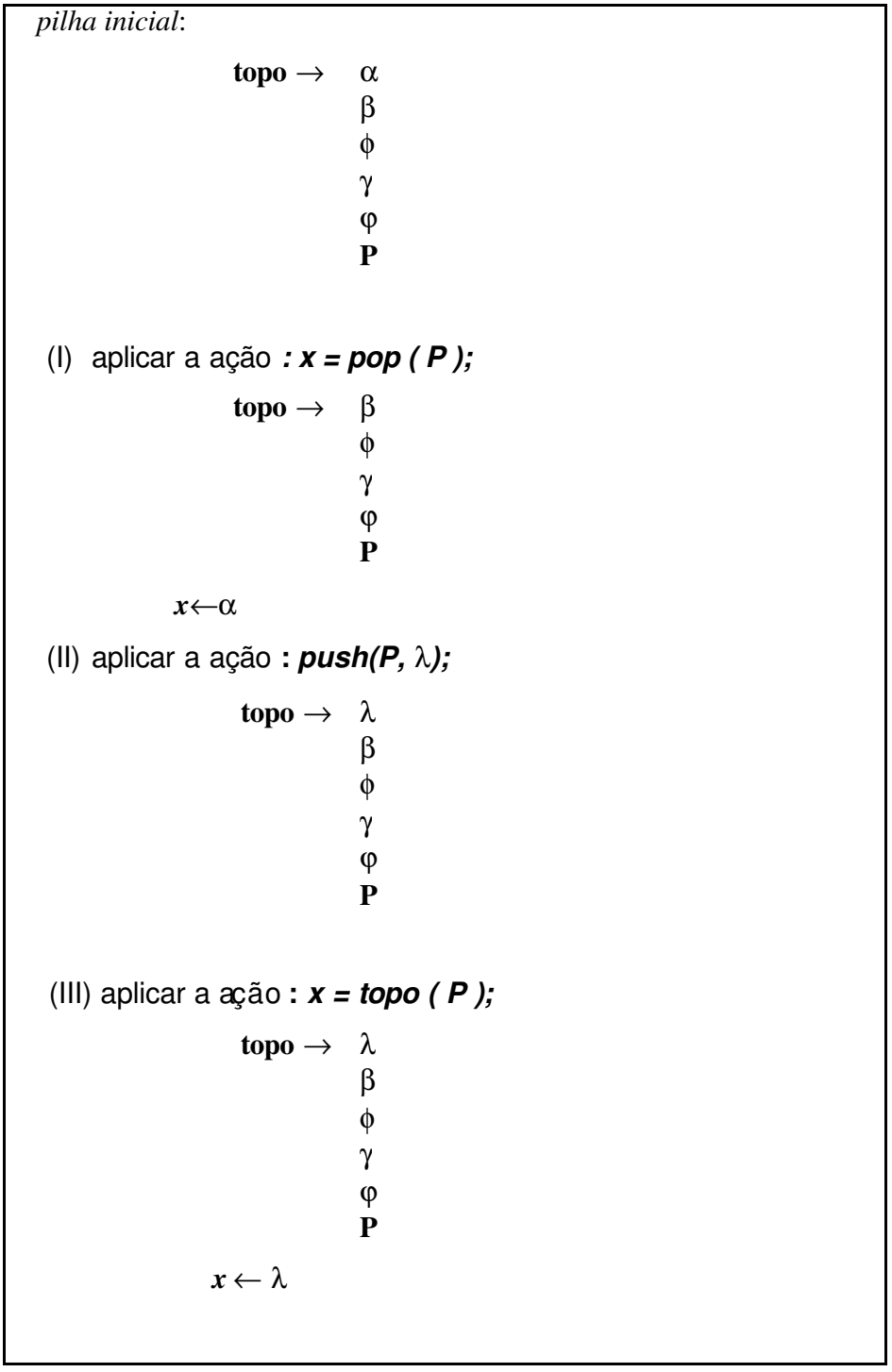


Figura 4.5: Representação da pilha P em uma seqüência de operações.

FILA:

Definição abstrata dos dados: fila é uma lista linear cujas inserções são efetuadas em um lado da estrutura, chamado *final da fila* e as eliminações no outro lado da estrutura, chamado *início da fila*.

Ações abstratas: são executadas em função das posições que caracterizam o início e o final da fila.

Iniciar:

ação : tornar a fila vazia, pronta para receber informações.
entrada : a fila que deverá ser transformada em vazia;
resultado : a fila pronta para receber as informações;
sintaxe : *init* (fila) $\rightarrow$ fila vazia.

Inserir:

ação : inserir uma informação no final da fila;
entrada : a fila que receberá a informação e a informação a ser inserida;
resultado : a fila alterada;
sintaxe : *inserir* (fila, valor) $\rightarrow$ fila modificada.

Eliminar:

ação : eliminar uma informação do início da fila;
entrada : a fila que sofrerá a eliminação;
resultado : a fila alterada e a informação retirada;
sintaxe : *eliminar* (fila) $\rightarrow$ fila modificada e elemento do início da fila.

Consultar:

ação : consultar uma informação contida no início da fila;
entrada : a fila para a consulta;
resultado : a informação consultada (fila inalterada);
sintaxe : *consultar* (fila) $\rightarrow$ cópia do elemento que está no início da fila.

Vazia:

ação : verificar se a fila está vazia ou não;
entrada : a fila a ser analisada;
resultado : verdadeiro (vazia) ou falso (não vazia);
sintaxe : *vazia* (fila) $\rightarrow$ verdadeiro ou falso.

As operações definidas para a fila se dividem em ações de construção e de inspeção:

ações de construção	ações de inspeção
init (fila)	consultar (fila)
inserir (fila, valor)	vazia (fila)
eliminar (fila)	

Considere as seguintes definições:

Q $\in$ fila; onde *Q*' representa a fila *Q* após execução da ação
max $\in$ inteiros; onde *max* : número máximo de elementos na fila
valor $\in$ tipo_do_elemento
b $\in$ { V, F}

REGRAS DE VALIDAÇÃO:

01. Erro – Overflow e Underflow
    a) eliminar (init (*Q*)) → Erro de “underflow”;
    b) consultar (init (*Q*)) → Erro de “underflow”
    c) inserir (*Q, valor*) → Erro de “overflow”, se e somente se a fila possuir *max* elementos;

AXIOMAS:

01. consultar (int (*Q*)) → Erro
02. consultar (inserir (*Q , valor*)) → *valor*, se vazia (*Q*) → V;
03. consultar (eliminar (*Q*))→ *valor*, onde valor é o segundo elemento da fila *Q*;
04. vazia (int (*Q*))→ V;
05. vazia (inserir (*Q*, *valor*))→ F;
06. vazia (eliminar (*Q*)) → V se *Q* possui apenas um elemento e F se *Q* possui mais de um elemento;

Representação Abstrata: a fila é representada destacando-se as extremidades que caracterizam o início e o final da fila e o nome dado a ela. Veja exemplo na Figura 4.6.

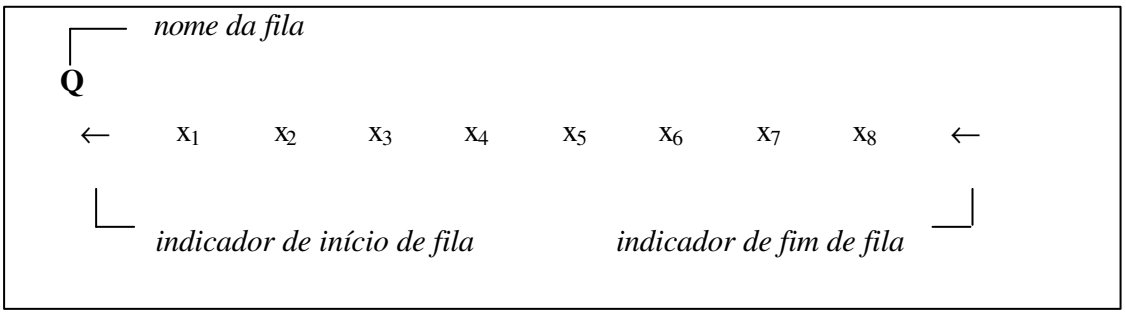


Figura 4.6: Representação Abstrata de uma Fila

A Figura 4.7 mostra um exemplo inicial da fila através de sua representação abstrata, contendo elementos constantes, representados por letras gregas. Uma seqüência de ações é executada sobre a fila e o resultado obtido da aplicação de cada uma delas é mostrado a seguir. A ações deste tipo devem corresponder às características de uma fila do mundo real.

Considere as seguintes declarações:

$Q \in$ fila;

$\alpha, \beta, \phi, \gamma, \varphi, \lambda, \theta$ (constantes) $\in$ tipo_do_elemento;

$b \in \{ V, F \}$;

$x \in$ tipo_do_elemento;

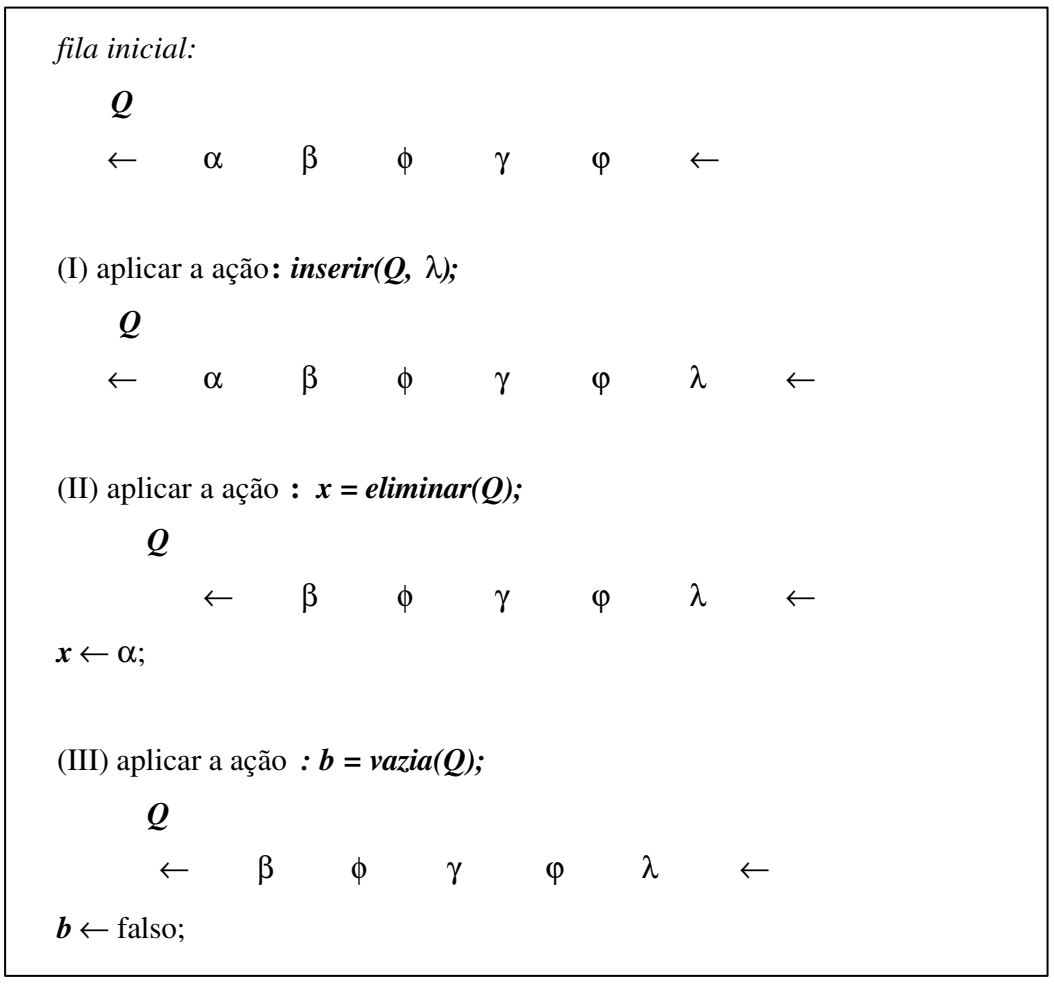


Figura 4.7: Representação da fila Q em uma seqüência de ações

CAPÍTULO V

DESCRIÇÃO DA ADTTool

Introdução

Ao construir funções que usam estruturas de dados como tipos de dados abstratos, o estudante necessita executá-las, sem ter que desenvolver as operações que manipulam as estruturas de dados.

A ferramenta ADTTool é um ambiente que permite ao estudante executar as funções que fazem uso das estruturas de dados básicas (listas lineares, filas e pilhas) como definidas no Capítulo IV e, através da interpretação da função construída, apresenta uma animação do algoritmo, mostrando seu comportamento lógico, usando como elemento da interface a representação visual abstrata definida para cada uma das estruturas de dados.

5.1 – ADTTool como Apoio ao Desenvolvimento

A camada da funcionalidade de um problema, apresentada na Figura 4.1 do capítulo anterior, refere-se à fase do desenvolvimento de um projeto onde o estudante deverá escrever a função necessária para atender as necessidades de um determinado problema. No caso, em se tratando de um curso inicial sobre estruturas de dados, as funções criadas para teste, inicialmente, estão restritas às três estruturas de dados básicas: lista linear, fila e pilha.

A fase que se segue é a da compilação da função construída e para isso é necessária a construção das operações. Isso significa que o estudante terá que decidir sobre o armazenamento das informações.

Essa decisão faz com que o estudante saia da percepção do abstrato, trabalhada até esse momento, para pensar em elementos concretos, que são as formas de armazenamento para a estrutura, tais como a representação por contigüidade física ou a encadeada (ligada). Nesta situação, antes mesmo de o estudante testar e verificar se a abstração empregada na construção de sua função está correta, ele passa a trabalhar com elementos concretos.

A ferramenta ADTTool foi criada para ser um ambiente de execução das funções, criadas pelos estudantes. Nela estão embutidas as definições das operações sobre as estruturas de dados, para que o estudante possa testar a abstração de sua função, sem ainda ter que se preocupar com a implementação das estruturas, isto é, a forma de representação interna das mesmas.

A Figura 5.1 apresenta um esquema da atuação da ferramenta, associada com a camada da funcionalidade descrita anteriormente. A figura está dividida em duas partes. Na primeira parte, estão representadas as fases que compreendem o processo usual de aprendizado de estruturas de dados, mesmo que apresentado como TDA. A primeira fase refere-se à teoria onde são passados para os estudantes os conceitos de uma nova estrutura de dados e as operações básicas sobre ela. Engloba a Camada da Funcionalidade do problema e a Camada de Abstração que envolve as operações. Como produto dessa primeira fase, tem-se a Função em C++ escrita pelo estudante. A fase seguinte refere-se à prática, isto é, compilar e executar a Função. Para isso é necessária a definição da forma de armazenamento da estrutura de dados e a construção das operações dessa estrutura. Essa escolha da forma de armazenamento e criação das operações constitui-se no Componente Concreto do processo.

Na segunda parte, estão representadas as mesmas fases da primeira porém com a inserção de uma fase intermediária, a Fase de Execução da abstração envolvida na função construída pelo estudante. Nesta fase intermediária está localizada a ferramenta ADTTool que, através da animação dos elementos da

função, demonstrará ao estudante se o resultado obtido pela representação visual está compatível com funcionalidade desejada no problema. Neste momento, não há necessidade de decidir sobre a forma de armazenamento, que pode ser qualquer uma, pois isto não deve interferir na funcionalidade do problema.

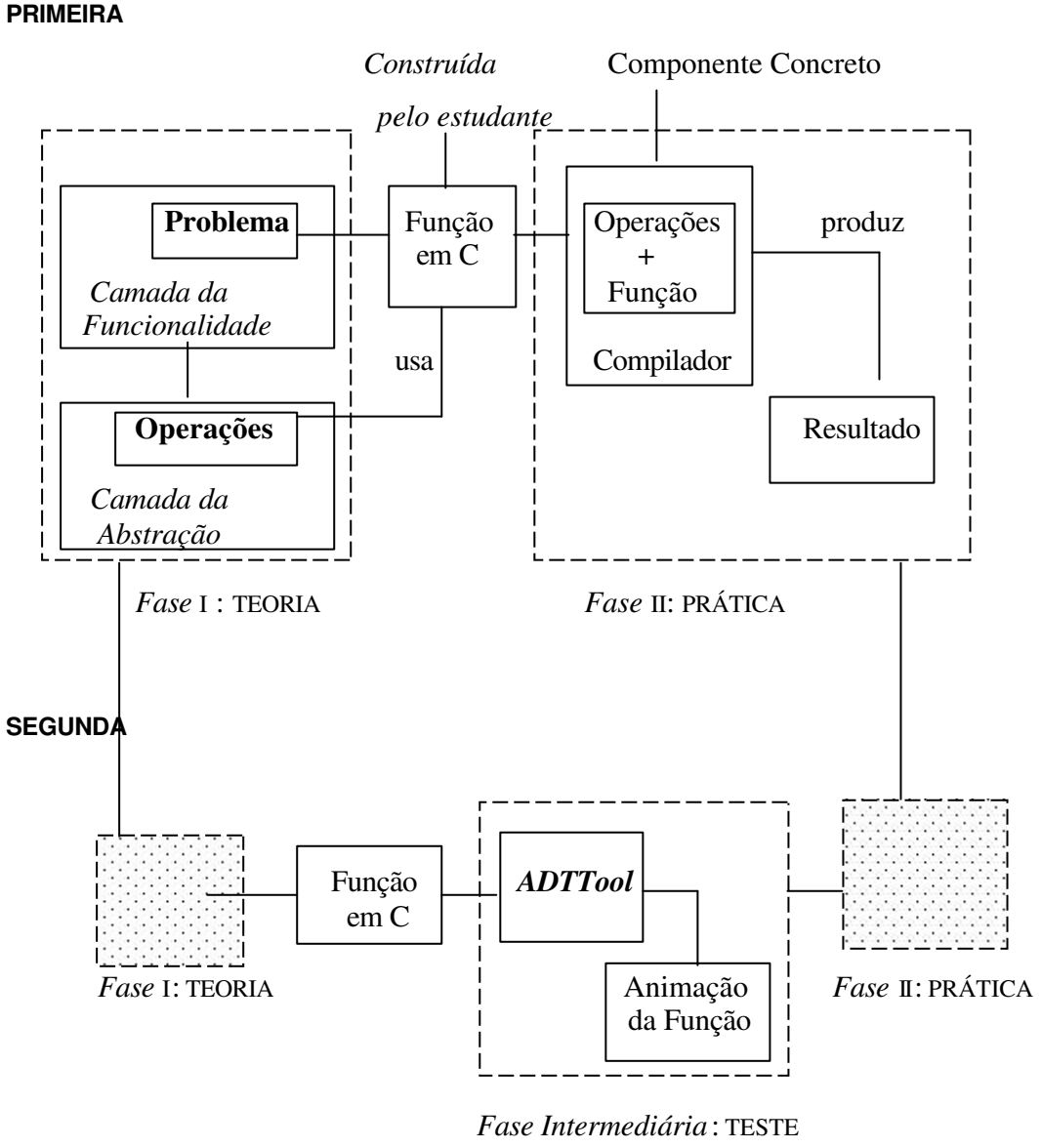


Figura 5.1 – Esquema de atuação da ADTTool

5.2 – Sobre o Desenvolvimento

A ADTTool interpreta uma função criada pelo estudante e gera uma animação das operações executadas sobre as estruturas de dados e as atribuições feitas às variáveis simples. A Figura 5.2 mostra um esquema da composição da ADTTool e das fases do seu desenvolvimento.

A ferramenta foi desenvolvida em três fases, sendo que a primeira tratou exclusivamente da elaboração do interpretador com relação à construção e o execução das funções que o compõem.

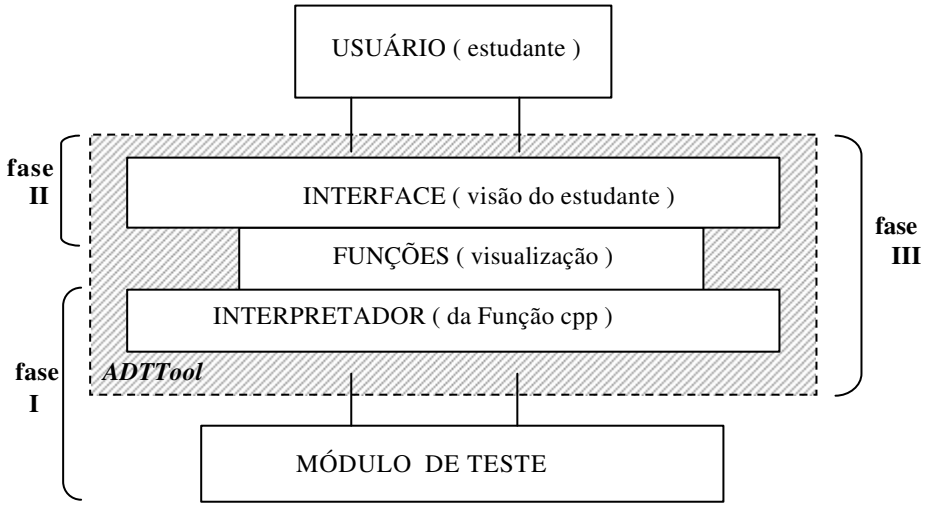


Figura 5.2 : Fases de Desenvolvimento de ADTTool

Um Módulo de Teste foi construído para testar a sintaxe da linguagem interpretada e o uso das estruturas de dados. O Interpretador agrega um total de 45 funções, sendo que, dessas, 38 tratam dos elementos relativos à sintaxe da linguagem e as demais para os exclusivos do uso dos tipos de dados referentes às três estruturas de dados: lista linear, pilha e fila. A listagem do programa fonte referente a ADTTool é apresentada no relatório técnico [ENG01].

Entre os testes executados sobre as estruturas de dados, estão os listados em Langsam [LAN96], como exercícios.

A segunda fase do desenvolvimento refere-se ao projeto da interface gráfica, representando a visão dos estudantes. Ela foi desenvolvida utilizando-se os recursos do Visual C++, mais especificamente as bibliotecas MFC (*Microsoft Foundation Classes*), pelas facilidades que oferece para a estruturação de aplicativos *Windows*. Nesta fase, 16 funções manipulam as ações associadas aos botões e ícones da interface.

A terceira fase do desenvolvimento refere-se à junção do interpretador com a interface gráfica. Essa ligação se deu através da criação de 17 funções responsáveis pela demonstração visual, na interface, dos resultados interpretados. Os resultados que geram a demonstração visual referem-se às ações abstratas aplicadas sobre as estruturas de dados básicas e às atribuições efetuadas sobre as variáveis simples. Essas funções demonstram as ações abstratas de acordo com a representação abstrata definida para cada uma das estruturas de dados, apresentadas no Capítulo IV.

A ferramenta é composta de 78 funções, com aproximadamente 4800 linhas de programa, que compreendem: o interpretador, os componentes da janela e a composição da interface, que faz a demonstração do algoritmo.

5.3 – O Interpretador

O interpretador foi construído na linguagem de programação C++, e apoia-se basicamente em cinco funções principais, que são:

1. procura dos *tokens* definidos como válidos para a implementação da ferramenta;
2. localização do inicio e final de cada bloco de comando;
3. interpretação e controle do fluxo lógico dentro de um bloco de comando;

4. análise das palavras-chave encontradas e chamada das funções que implementam cada uma das palavras-chave;
5. avaliação das expressões aritméticas e lógicas.

As três estruturas de dados básicas: pilha, fila e lista linear, usadas no desenvolvimento da ferramenta, foram desenvolvidas considerando-se a abordagem orientada a objetos, permitindo-se, assim, a escolha para conteúdos do tipo *int*, *float* ou *char*. Entretanto, nesta primeira versão da ferramenta ADTTool, não está implementada a possibilidade de escolha do tipo, nem a entrada de dados. Todos os exemplos são demonstrados utilizando-se o tipo inteiro. O uso de um tipo fixo inicialmente não compromete a compreensão e a observação da abstração dos tipos de dados uma vez que as ações abstratas são aplicadas a qualquer conjunto de dados. A incorporação futura desta escolha é importante, pois acrescentará ao aprendizado do estudante que o comportamento das estruturas de dados resultante das ações será o mesmo independentemente de o conteúdo de uma lista linear, por exemplo, ser de inteiros ou de reais.

5.4 - Funções de Visualização

A animação de um algoritmo abordada pela ADTTool refere-se ao fluxo de dados dentro da função. A demonstração do fluxo dos dados é feita por um conjunto de funções de visualização ligadas às operações definidas para as estruturas de dados. Cada uma das operações possui um conjunto de funções de visualização correspondente. Assim sendo, cada operação de uma estrutura de dado, que é encontrada pelo interpretador, a ação correspondente é efetuada na estrutura interna da ferramenta e as funções de visualização são executadas.

Além das operações ligadas às estruturas de dados, a atribuição de um novo valor às variáveis simples do problema também gera alterações na interface. Assim sendo, existe uma função de visualização para a operação de atribuição.

5.5 - Descrição da Interface

A interface da ferramenta apresenta uma linguagem visual específica criada especialmente para demonstrar a abstração das estruturas de dados através das representações abstratas criadas para cada uma delas.

A interface gráfica é composta de uma janela padrão *Windows*, com os itens básicos, tais como, moldura da janela; barra de título dotada da capacidade de controle para ser movida, com botão para fechamento da mesma e o título; a área cliente que contém o espaço de visualização. Os componentes barra de rolagem, barra de *status* e barra de menu não foram incorporados à ferramenta e a área cliente, que representa a ferramenta, é composta de elementos visuais divididos em cinco áreas de visualização e sete botões.

ÁREA DE VISUALIZAÇÃO

A área de visualização é composta de cinco áreas fixas, isto é, não variam de acordo com a função executada. Assim sendo, a visão inicial que o estudante terá da interface gráfica será sempre a mesma. Essa representação com elementos fixos facilita a localização na interface, da parte dos estudantes, dos elementos da função criada por eles. Em função disto, cada espaço possui um limite de representação para cada elemento.

A Figura 5.3 mostra a Tela Inicial da interface gráfica da ADTTool com as cinco áreas destacadas, que são:

- ÁREA DE VARIÁVEIS: área para visualização do conteúdo das variáveis simples (A) e do respectivo nome;

- ÁREA DE PILHAS: área de visualização da estrutura de pilha (B).

- ÁREA DE LISTAS E FILAS: área de visualização das estruturas : fila e lista(C).

- ÁREA DE LISTAGEM: área para a visualização da listagem da função que será analisada pela ferramenta (D);

- ÁREA PARA NOME DO ARQUIVO: área para visualização do nome (com o caminho) do arquivo, com a função que será analisada pela ferramenta (E).

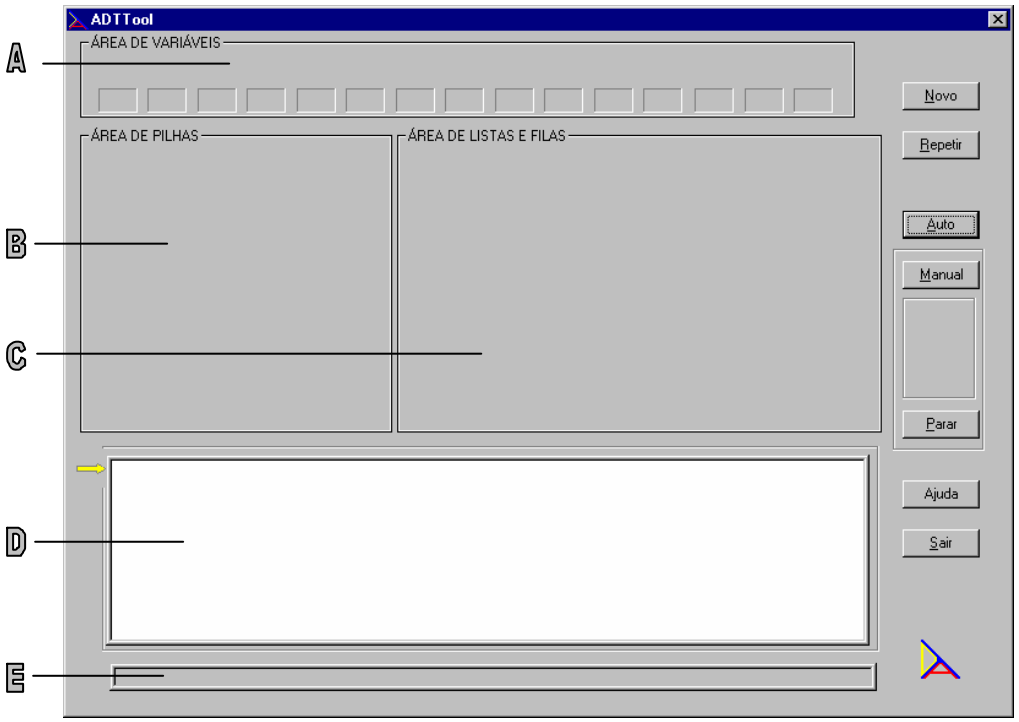


Figura 5.3 : Áreas de Visualização da Interface de ADTTool

ÁREA DE VARIÁVEIS: todas as variáveis simples, definidas na função, serão visualizadas nesta área. Sempre que ocorrer a atribuição de um valor a uma variável, esse valor será visualizado. As variáveis colocadas na lista de parâmetros serão iniciadas com um valor aleatório, maior ou igual a cinco. O espaço está limitado para no máximo 14 variáveis, as 14 primeiras que aparecerem na função. As demais não serão visualizadas.

ÁREA DE PILHAS: o espaço está limitado para a visualização de, no máximo, três pilhas. As demais não serão visualizadas. As variáveis do tipo pilha que fizerem parte da lista de parâmetros da função serão iniciadas com, no mínimo, cinco valores inteiros aleatórios, podendo haver repetições ou não. Caso o exemplo mostrado não seja adequado para a função a ser interpretada pela ADTTool, basta escolher o botão Repetir, que um novo conjunto de valores será apresentado.

ÁREA DE LISTAS E FILAS: o espaço está limitado para a visualização de no máximo quatro das duas estruturas Fila e/ou Lista, podendo haver qualquer combinação delas, tais como: 4 filas , 4 listas, 2 listas e 2 filas, 3 listas e 1 fila, etc.

ÁREA DE LISTAGEM: o espaço mostrará a função escolhida e aberta através da escolha do botão *Novo*. A seta localizada no canto superior esquerdo indica o comando que está sendo interpretado pela ADTTool naquele momento. A visualização dessa operação é possível apenas quando escolhido o modo de exibição *Manual*. Para funções muito pequenas, com poucas linhas de comando, toda a função é visualizada, em qualquer momento da interpretação da mesma.

ÁREA PARA NOME DO ARQUIVO: o espaço mostrará o nome e o caminho do arquivo que contém a função escolhida para ser interpretada pela ADTTool.

BOTÕES

A interface possui sete botões que representam elementos da interface cujas ações, associadas a eles, produzem modificações nas áreas de visualização, representando a animação. A Figura 5.4 mostra a interface gráfica inicial da ADTTool com os seguintes botões:

Novo (I): usado para a escolha e abertura de um arquivo com a função a ser interpretada pela ADTTool;

Repetir (II): usado para repetir a interpretação da função aberta, podendo ser escolhido, posteriormente, um dos dois botões: Auto ou Manual;

Auto (III): usado para execução rápida e automática da função;

Manual (IV): usado para a execução da função passo a passo

Parar (V): usado quando escolhido o modo Manual e, antes de encerrá-lo, deseja-se interrompê-lo;

Ajuda (VI): para obter ajuda sobre o modo de execução e detalhes sobre a ADTTool;

Sair (VII): para sair da ADTTool.

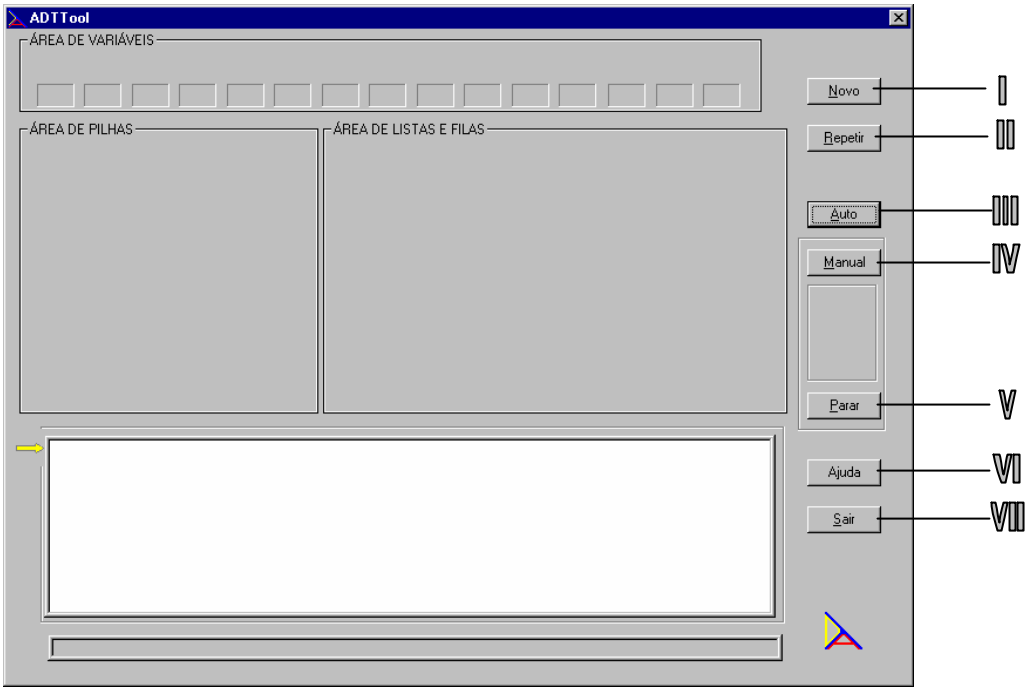


Figura 5.4 : Botões da Interface da ADTTool

5.6 – Elementos para a Construção das Funções a serem Animadas

Nesta primeira versão da ferramenta não foram implementados todos os recursos da linguagem C++, uma vez que a finalidade é testar programas simples que utilizam estruturas de dados como TDA. Não se trata de um curso para ensinar a programar em C++.

Elementos da linguagem C++ aceitos nessa primeira versão são:

1 - Palavras-chave: *char, do, else, float, for, if, int, void, while e return* **;**

2 - Tipos de dados: *int, float, char*, pilha, fila e lista;

- para as variáveis simples: *int, float e char*;
- para as estruturas: nesta versão apenas para o tipo inteiro.

3 - Operações:

- comuns às três estruturas:

  void init (< pilha *p* > | < fila *q* > | < lista *l* >)

  int vazia(< pilha *p* > | < fila *q* > | < lista *l* >)

- da estrutura de dados pilha

  void push(pilha *p*, *int x*)

  int pop(pilha *p*)

  int topo(pilha *p*)

- da estrutura de dados fila

  void inserir (fila q, *int x*)

  int eliminar (fila q)

  int consultar (fila *q*);

- da estrutura de dados lista linear

  void inserir (lista *l*, *int x* , *int pos*)

  int eliminar (lista l , *int* pos)

  int consultar (lista l , *int* pos)

  void alterar (lista l, *int* x, *int* pos)

  int compr (lista l)

4 - Usos implementados dos comandos *for, while, do-while e if* : não há restrições.

Dentre os recursos da linguagem que não foram implementados nesta versão da ADTTool estão: ponteiro, arranjos, *struct*, operadores sobrecarregados em expressões aritméticas, recursividade. Além das palavras chave: *auto, break, case, const, continue, default, double, enum, extern, goto, long, register, short, signed, sizeof, static, struct, switch, typedef, union, unsigened, volatile.*

CAPÍTULO VI

CONSIDERAÇÕES SOBRE A APLICAÇÃO DA ADTTool

Introdução

Este capítulo apresenta um exemplo de como utilizar a ADTTool e algumas considerações sobre as observações colhidas da apresentação piloto feita com alunos dos cursos de graduação de Engenharia de Computação e Análise de Sistemas.

6.1 – Como Executar

A partir de um editor do compilador C/C++ deve ser criada uma função *void*. A função deverá ser não-recursiva e não possuir chamada de outras funções. Na ADTTool será feita a execução de funções isoladas, isto é, uma de cada vez. A função pode ter uma lista de parâmetros.

Todas as variáveis colocadas na lista de parâmetros receberão como valores iniciais números aleatórios. As variáveis da lista de parâmetros que, no corpo da função interpretada, sofram uma atribuição terão os valores iniciais substituídos pelas atribuições.

Essas funções construídas para serem testadas pela ADTTool devem ser especificas para o ensino de estruturas de dados e, portanto, devem fazer uso das operações das três estruturas de dados: pilha, fila e lista linear, como foram definidas no capítulo anterior.

As funções a serem interpretadas pela ADTTool devem ser escritas usando-se C++, observando-se os recursos implementados e apresentados no Capítulo V, e terem sido compiladas para a eliminação de erros de sintaxe.

Para a execução da ferramenta ADTTool, é necessário que a função esteja sem erros de sintaxe, pois nem todos os erros são detectados na ferramenta, uma vez que o objetivo não é o mesmo de um compilador C++.

Para que o compilador reconheça as operações definidas para as estruturas, deve-se usar um arquivo *header* como descrito na Figura 6.1, onde (a) refere-se à estrutura de pilha; (b), à de fila; e (c), à de lista linear.

```
typedef elemento;
typedef pilha;
elemento pop(pilha);
elemento topo(pilha);
void push(pilha, elemento);
void init(pilha);
int vazia(pilha);
```

(a)

```
typedef fila;
elemento eliminar(fila);
void inserir(fila, elemento);
elemento consultar (fila);
int vazia(fila);
void init(fila);
```

(b)

```
typedef lista;
elemento eliminar(lista, int);
void inserir(lista, elemento, int);
elemento consultar (lista, int);
int vazia(lista);
void init(lista);
int compr(lista);
```

(c)

Figura 6.1: Arquivo *header*

Após a compilação, a função estará pronta para ser interpretada pela ADTTool, cuja Tela Inicial é mostrada na Figura 6.2.

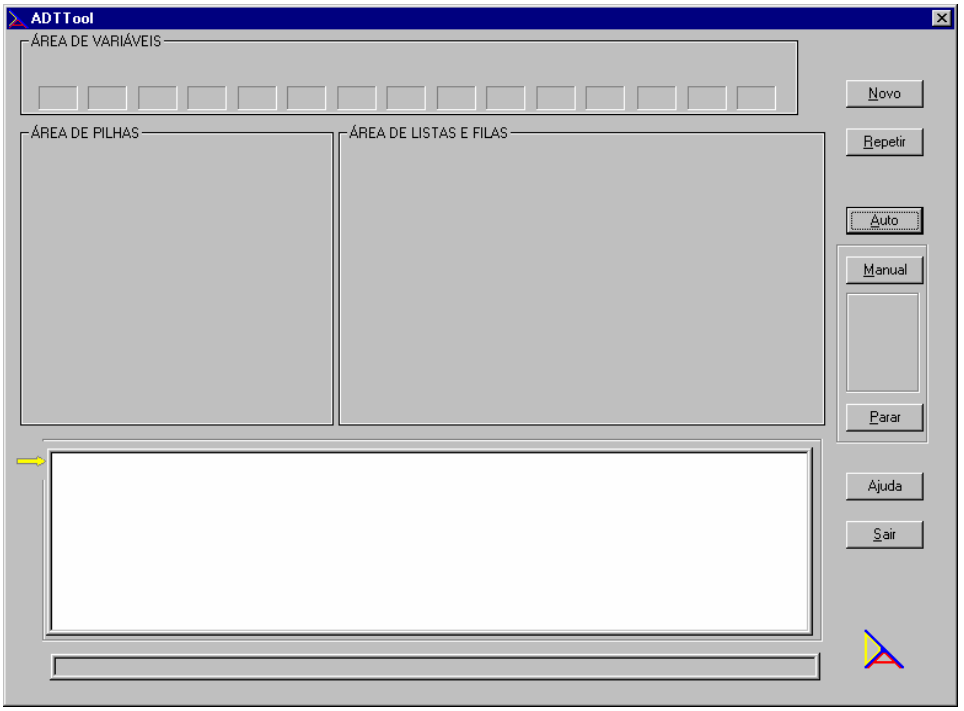


Figura 6.2: Tela Inicial da ADTTool

As opções de escolha dos botões possuem uma relação lógica de dependência para que a função possa ser executada sem a ocorrência de avisos de erros sobre a escolha.

Essa relação pode ser visualizada através de um dígrafo, mostrado na Figura 6.3. Cada um dos botões representa um nó do dígrafo: (2) Novo, (3) Auto, (4) Manual, (5) Repetir, (6) Parar e (7) Sair.

A Tela Inicial, como ponto de partida da interface, é representada pelo nó de número um.

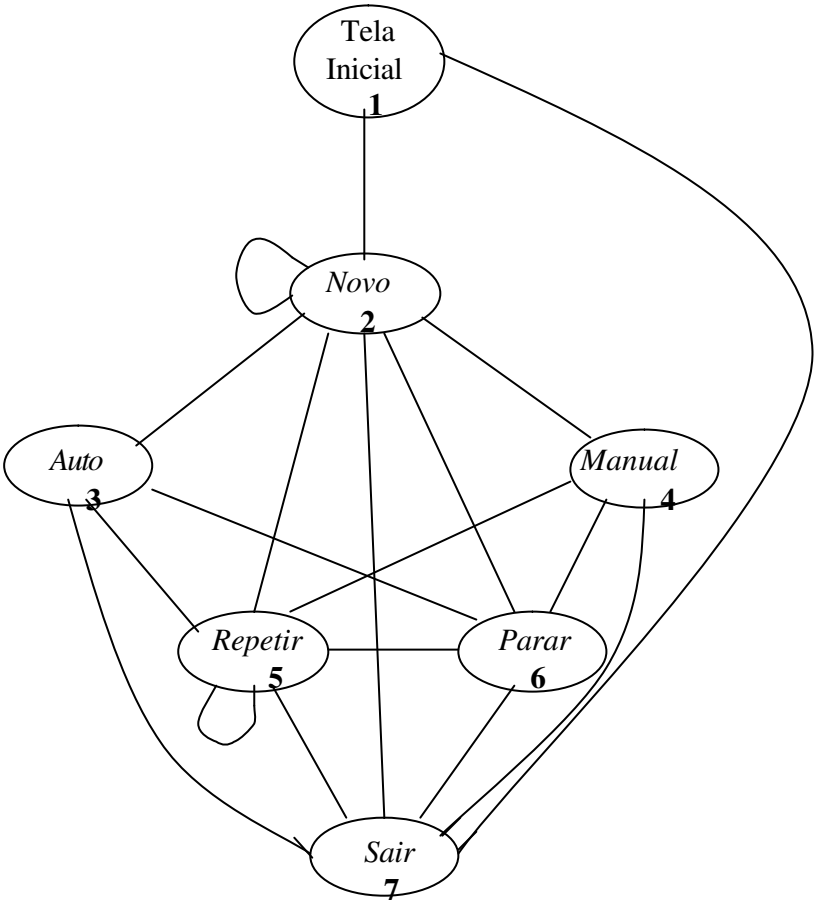


Figura 6.3 – Relação de dependência entre as escolhas dos Botões da Interface

Partindo-se da Tela Inicial (nó um) até a primeira execução da animação de uma função através da escolha do botão *Auto*, existe um único caminho, sem repetições de escolha, que é dado por:

```
Tela Inicial --> Novo --> Auto
```

Da mesma forma, se a escolha para a primeira execução da animação da função for através do botão *Manual*, existe um único caminho sem repetições de escolha, que é dado por:

```
Tela Inicial --> Novo --> Manual
```

O conjunto de possibilidades de caminhos, representando escolhas, dentro de uma seqüência lógica que produza resultados esperados, é apresentado no Anexo II. Esse conjunto representa apenas os caminhos sem repetições.

Considerando os dois arcos que representam ciclos, referentes aos botões Novo e Repetir, que são <Novo, Novo> e <Repetir, Repetir>, pode-se obter uma grande quantidade de caminhos de escolha dos botões.

O ciclo < Novo, Novo> de escolha refere-se à possibilidade de o estudante mudar a função escolhida sem ter efetuado a animação da anterior.

O ciclo <Repetir, Repetir> de escolha refere-se à possibilidade do estudante desejar um novo conjunto de valores para a inicialização das variáveis. Esse ciclo pode se repetir até que o conjunto apresentado atenda às necessidades do estudante. A seguir são apresentadas figuras com as telas da ADTTool, referentes ao caminho:

```
Novo --> Auto     --> Repetir --> Manual
```

Escolhendo-se o botão *Novo*, será aberta a janela para a escolha da função a ser executada. No exemplo foi escolhida a função *prog60.cpp (*Figura 6.4).

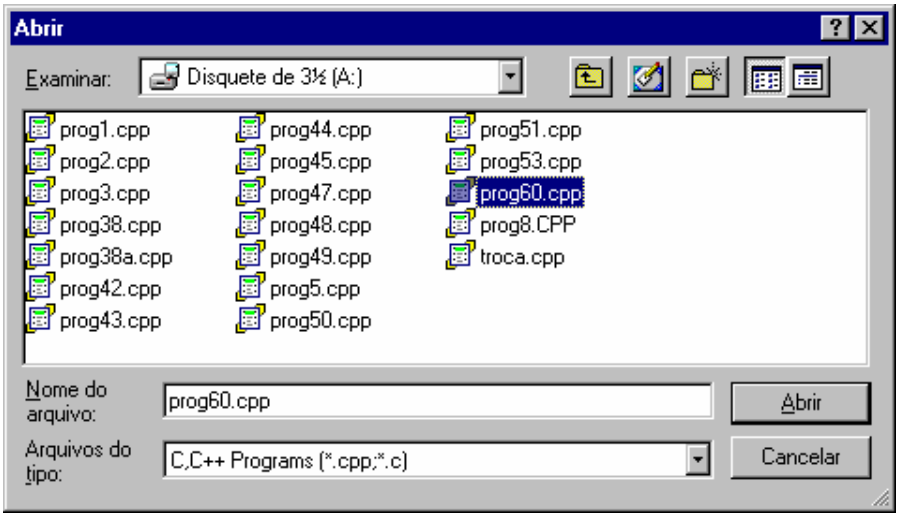


Figura 6.4 – Tela para escolha da função a ser interpretada pela ADTTool

A Figura 6.5 apresenta o resultado da escolha da função. A função escolhida, prog60.cpp, tem como funcionalidade empilhar apenas os elementos pares de uma lista linear.

A partir da escolha da função a ser animada, a ADTTool apresentará na ÁREA PARA NOME DO ARQUIVO o caminho e o nome do arquivo escolhido; na ÁREA DE LISTAGEM, a listagem da função escolhida e na ÁREA DE LISTAS E FILAS, valores aleatórios iniciais para a lista linear. No exemplo a lista linear inicial gerada possui seis elementos.

As demais áreas de visualização não recebem valores iniciais uma vez que essa inicialização somente ocorre para as variáveis que constam da lista de parâmetros. No exemplo, a função possui como parâmetro apenas uma lista linear.

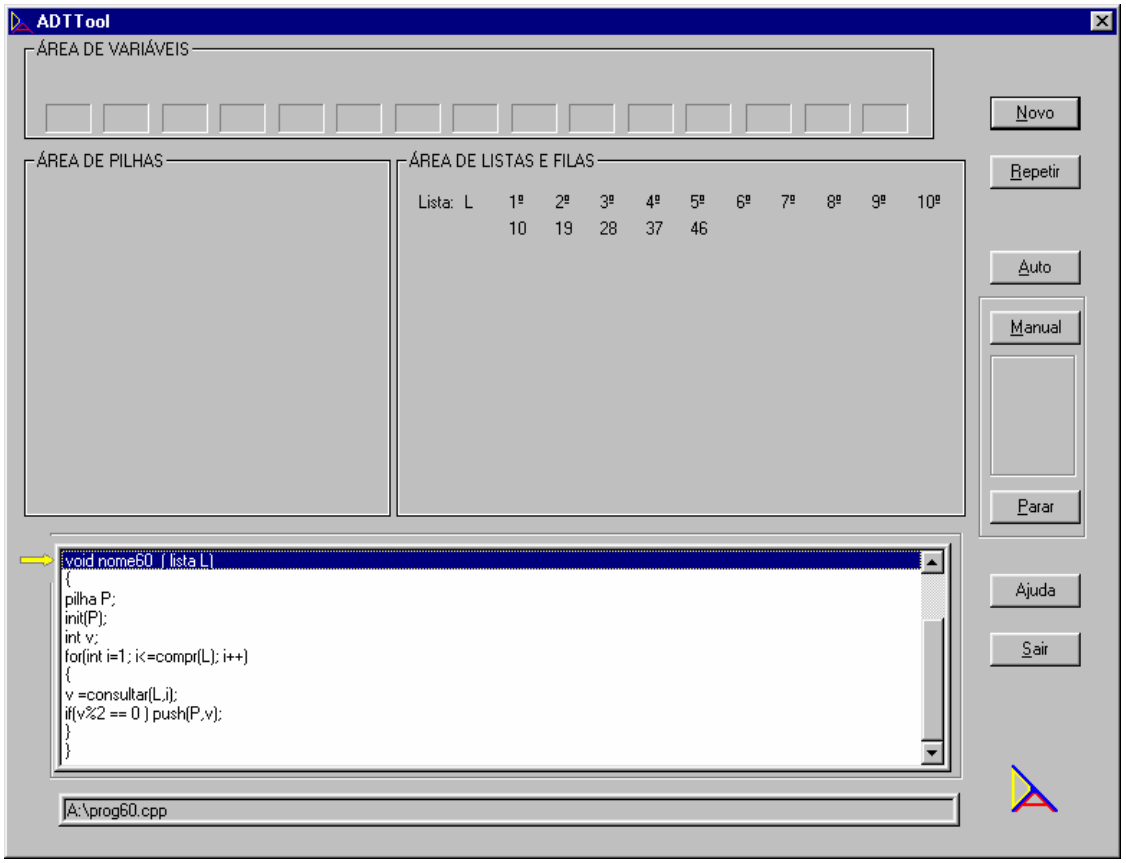


Figura 6.5: Tela com a escolha do arquivo prog60.cpp

Caso o estudante queira outro conjunto de dados iniciais para a lista, pode escolher o botão *Repetir*, quantas vezes desejar. Esta escolha refere-se ao ciclo: `<Repetir, Repetir>`. A Figura 6.6 apresenta a tela resultante da escolha do botão *Repetir*, com um novo conjunto de dados para a lista linear.

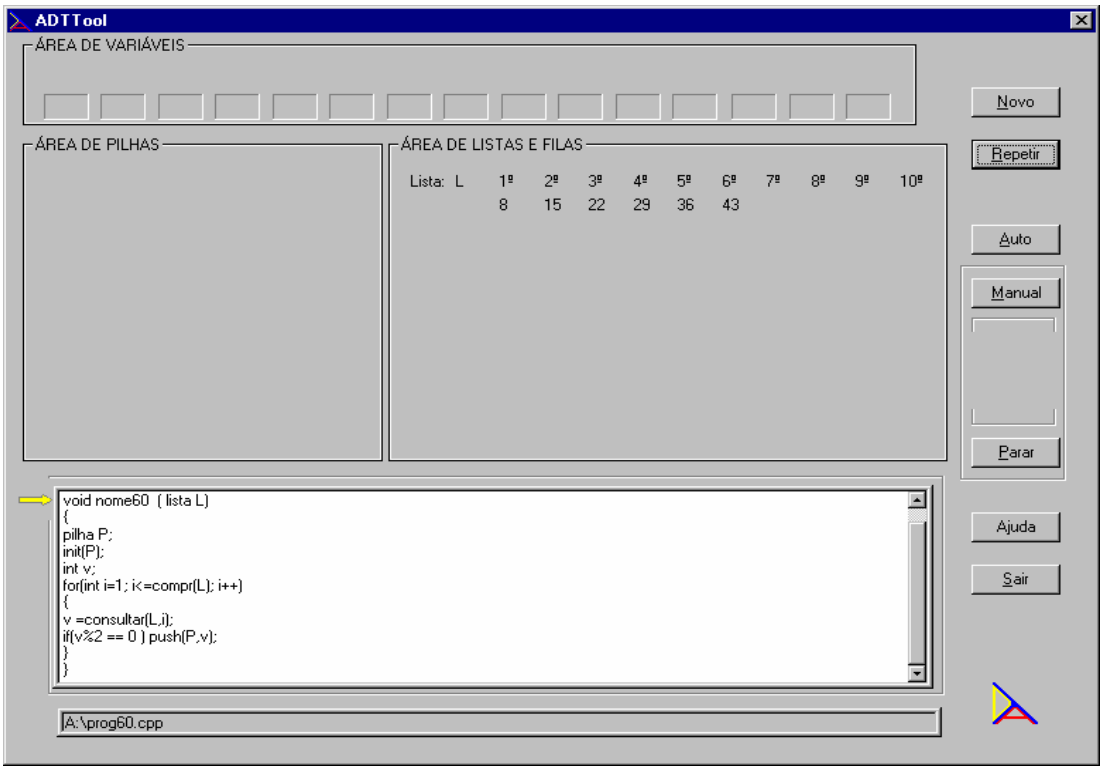


Figura 6.6: Tela resultante da escolha do arquivo prog60.cpp

A função escolhida além da lista linear passada como parâmetro possui uma pilha e as variáveis simples *v* e *i*.

Para a visualização da ação das operações das três estruturas utilizadas: lista linear e pilha, pode-se escolher a execução *automática* ou a *manual*. A automática executará a função toda, rapidamente, visualizando-se apenas o estado final das estruturas. A Figura 6.7 mostra o resultado após a escolha do botão *Auto*.

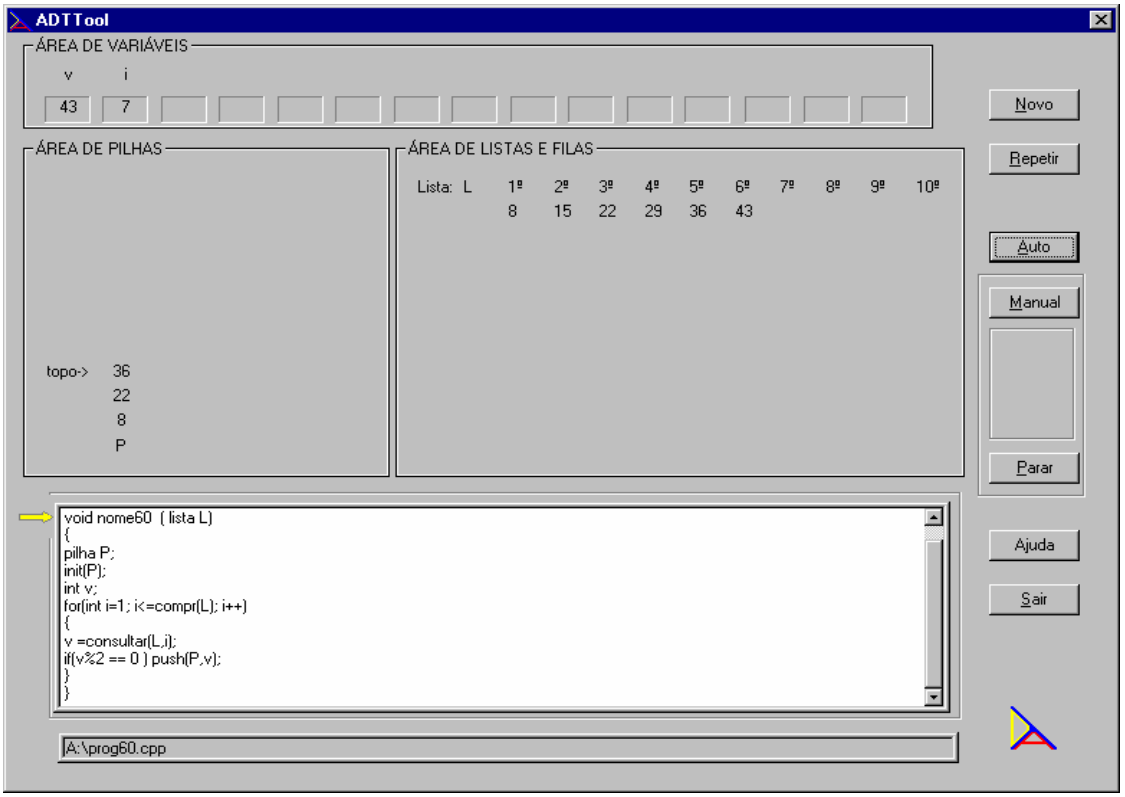


Figura 6.7: Tela após escolha do botão Auto

Pode-se observar que a pilha *P* passou a ter todos os elementos pares que a lista possui, tendo como elemento de topo o número *36*. Este número refere-se ao último número par que a lista linear contém. A variável *v* está com o último elemento da lista linear, pois foi a última operação de consulta executada. A variável *i* está com o valor *7* referente ao valor do comprimento da lista linear mais uma unidade, pois trata-se da condição de parada do comando *for*.

Após a escolha do botão *Auto* ou do *Manual*, as opções de escolha são:

```
Auto --> Novo
Auto --> Repetir
Auto --> Sair
```

A escolha do botão *Novo* deve ser feita se o estudante desejar testar uma nova função; a escolha do botão *Sair* significa que deseja sair do ambiente de execução.

Escolhido o botão *Repetir*, a tela mostrará um novo conjunto de valores iniciais para as variáveis que são parâmetros. A Figura 6.8 mostra a tela após a escolha do botão *Repetir*, onde aparece a mesma função escolhida anteriormente, cuja lista linear apresenta-se com novo conjunto de dados e as variáveis *v* e *i* sem valor, pois ainda não pertencem à lista de parâmetros da função.

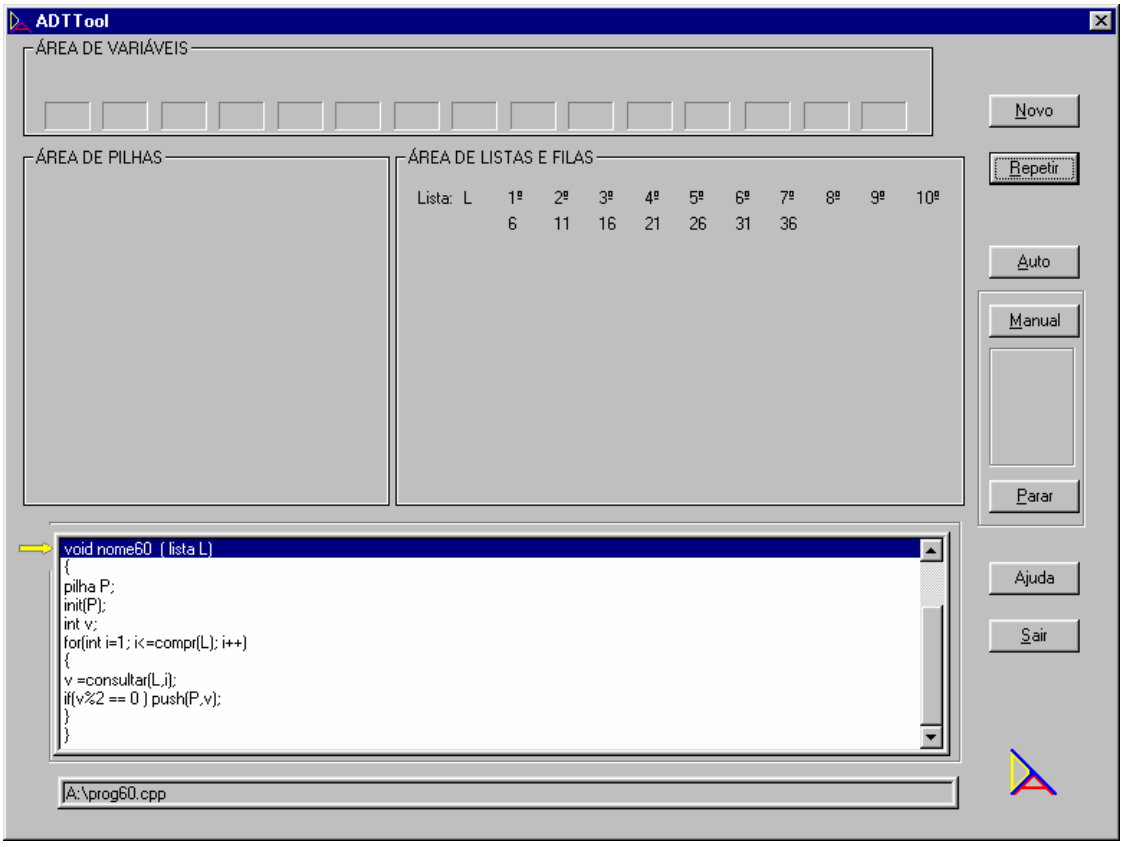


Figura 6.8 – Tela da escolha do Botão Repetir

A partir deste ponto, pode-se escolher um dos botões: *Auto*, *Manual*, *Novo, Sair* e *Repetir (ciclo)*, conforme os caminhos:

```
Repetir --> Auto
Repetir --> Manual
Repetir --> Novo
Repetir --> Sair
```

Escolhido o botão *Manual*, será mostrada a indicação de controle do "passo a passo", logo abaixo do botão *Manual*, como apresentado na Figura 6.9. Este controle do "passo a passo" será feito através do uso da "seta direita", possibilitando o controle da demonstração da função pelo estudante.

Figura 6.9 – Parte da tela mostrando a indicação da execução passo a passo

A visualização se dará em função da existência das operações sobre as estruturas de dados e da atribuição de valores novos às variáveis da função, uma vez que as funções de visualização estão associadas a essas operações. A Figura 6.10 mostra a tela com a escolha do botão *Manual* e uma indicação da localização do elemento para o passo a passo.

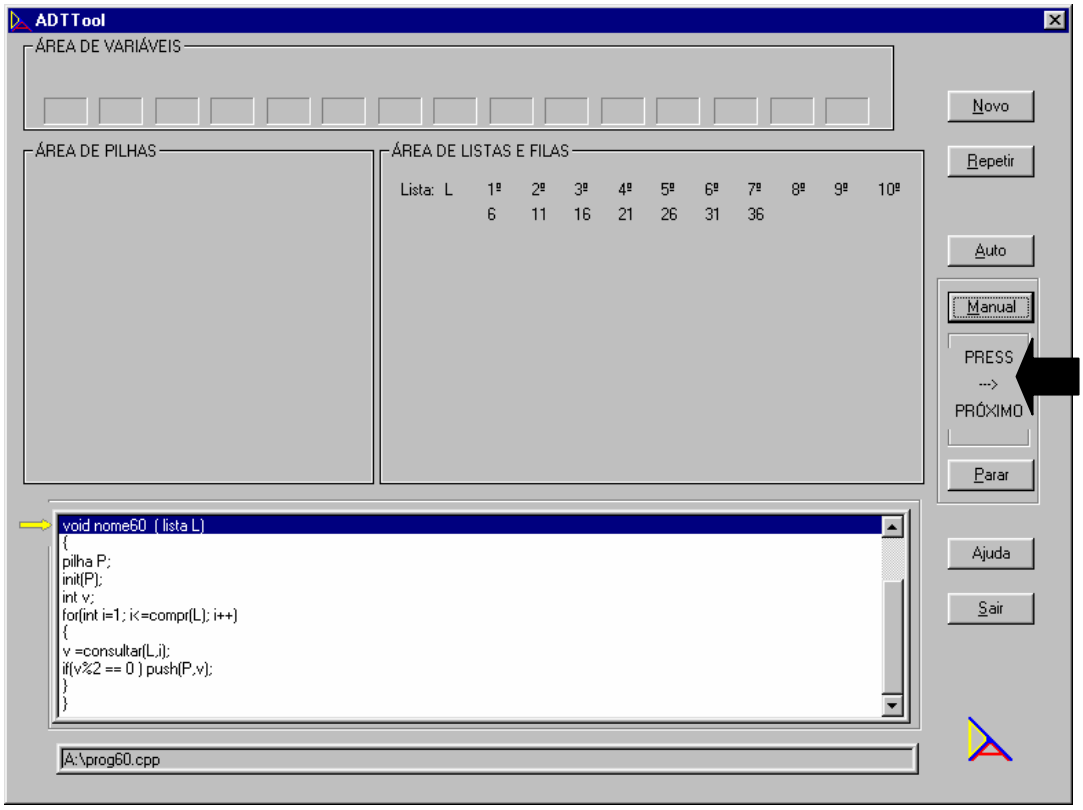


Figura 6.10: Tela depois de escolhido botão Manual

Após a primeira pressão da tecla “seta direita”, o interpretador executará todas as instruções até que encontre uma ação referente à atribuição, expressão aritmética, expressão lógica, operação sobre uma das estruturas de dados. Além das operações de definição de variáveis, estas ações são as que geram visualização. Assim sendo, o controle do processo manual está diretamente ligado às funções de visualização.

A Figura 6.11 apresenta a tela referente ao resultado obtido após a primeira vez que a tecla “seta direita” é pressionada, onde se pode observar o aparecimento do nome da variável que representa a pilha e as variáveis *v* e *i*.

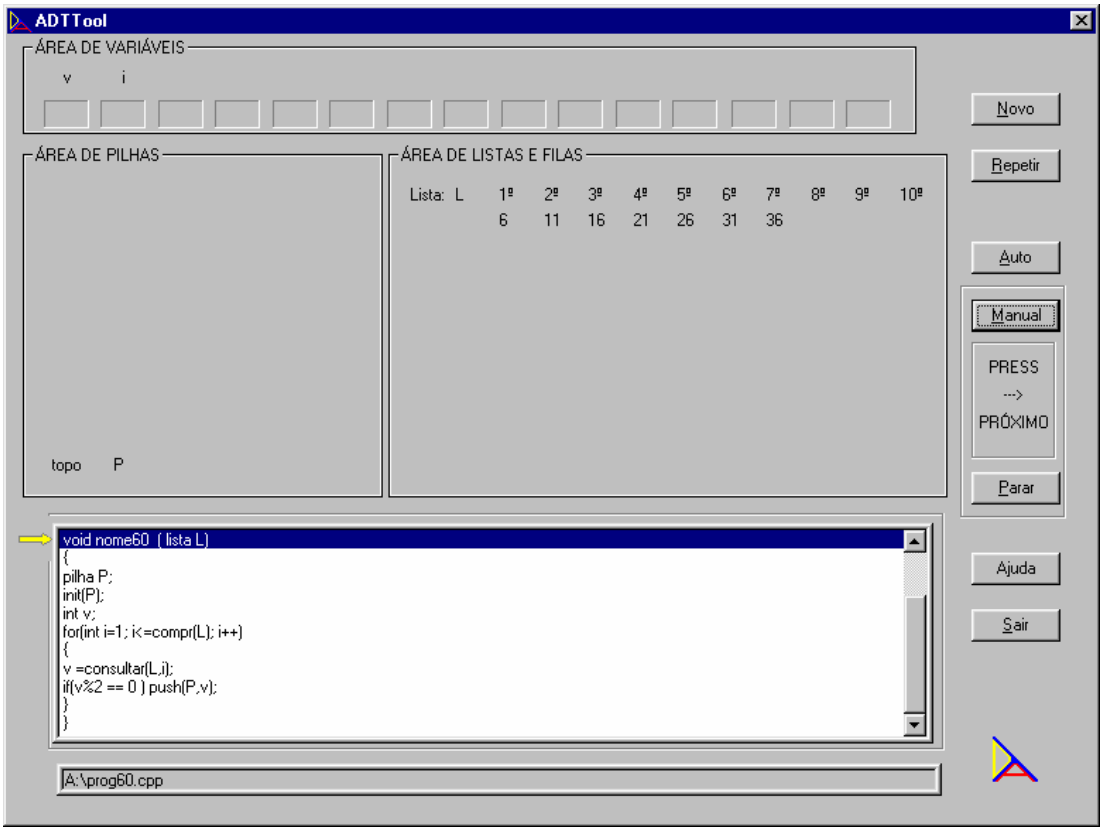


Figura 6.11: Tela depois de pressionada a tecla "seta direita" pela primeira vez

Depois de pressionada algumas vezes a tecla "seta direita", os elementos da interface apresentarão a seguinte situação:

- a variável *i* = 4: significa que a função está no passo onde será analisada a 4ª posição da lista (*i* = 4).;
- a variável *v* = 21: significa que a operação **consultar(*L,i*),** sobre a lista linear, já foi executada e a variável *v* já recebeu o conteúdo da 4ª posição;
- a pilha está com as informações **6** e **16**, que são os números pares da lista linear que já foram analisados e empilhados em *P*.

A Figura 6.12 mostra a tela que representa a situação descrita acima.

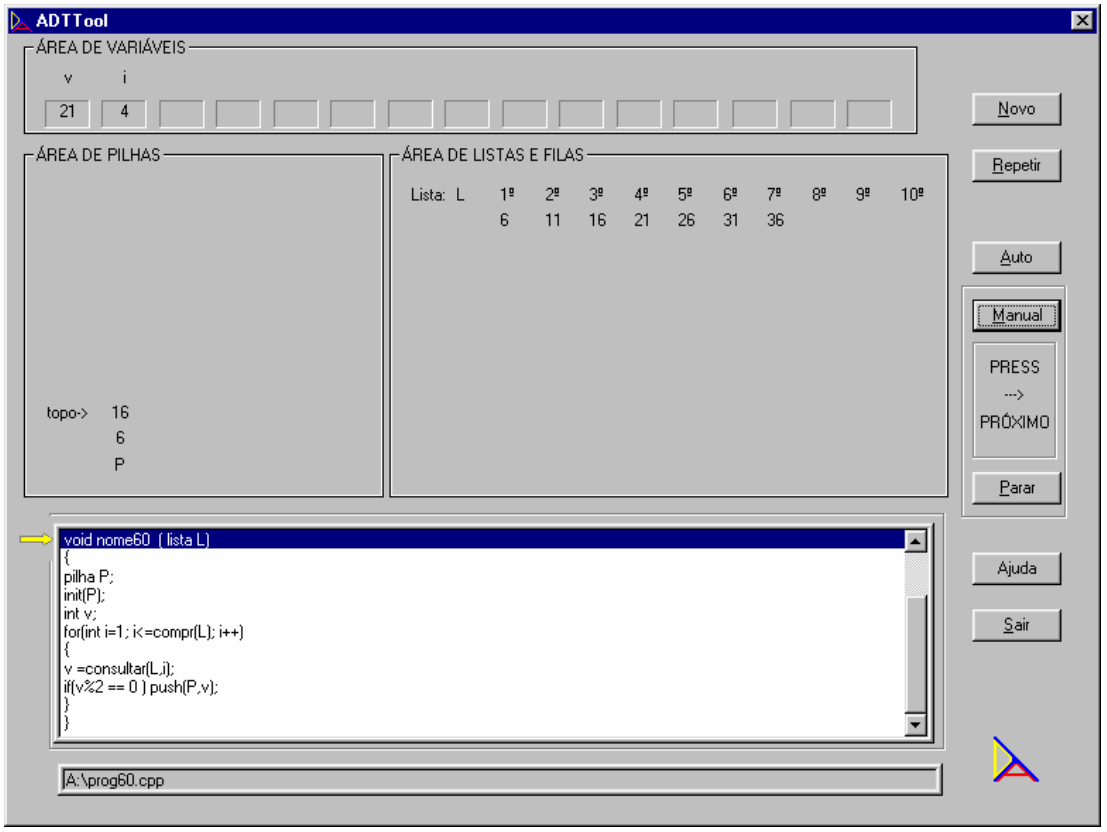


Figura 6.12: Tela depois de pressionada várias vezes a tecla “seta direita”

CAPÍTULO VII

CONSIDERAÇÕES SOBRE A APRESENTAÇÃO DA FERRAMENTA

Introdução

Este capítulo apresenta algumas considerações sobre as observações colhidas da apresentação piloto feita com alunos dos cursos de graduação de Engenharia de Computação e Análise de Sistemas.

7.1 – Método

A abordagem escolhida foi uma pesquisa de opinião a respeito da ferramenta ADTTool e sua aplicação na execução das funções que empregam estruturas de dados como tipo de dado abstrato.

O objetivo desta pesquisa foi buscar subsídios que pudessem orientar melhorias e propostas futuras para a ferramenta.

É importante destacar que os estudantes escolhidos não formam um conjunto representativo do universo e portanto não é possível generalizar os resultados e impressões colhidas na pesquisa de opinião.

7.2 – Sujeito

O sujeito adequado a participar desta fase de pesquisa de opinião deveria ser um estudante de um dos cursos de computação da PUC-Campinas, que já tivesse cursado e obtido aprovação na disciplina de Estruturas de Dados. Além

disso, o estudante deveria ter aprendido estruturas de dados como tipo de dado abstrato.

Após alguns convites, no total de 23, seis estudantes se prontificaram a examinar a ferramenta e apresentarem seus comentários sobre ela. Outros cinco, que não foram convidados, se ofereceram para participar, entretanto apenas um deles pode participar pois os demais ainda não haviam obtido aprovação na disciplina de Estruturas de Dados.

7.3 – Procedimento

Os estudantes foram convidados pessoalmente ou através de correspondência eletrônica. Do convite constou uma explicação sobre o desenvolvimento e os objetivos da ferramenta, o objetivo da pesquisa e a contribuição esperada por parte deles.

Após vários contatos definiu-se uma data para a demonstração da ferramenta a ser realizada no Laboratório de Informática da PUC Campinas. A demonstração foi iniciada com uma breve recordação sobre a abordagem de Estrutura de Dado como tipo de dado abstrato, procurando destacar a formalização e a representação visual associada a cada uma delas.

Esta primeira explanação, além de recordar a matéria, teve como objetivo estabelecer um padrão na linguagem envolvida com a ferramenta, embora ser esta já conhecida dos estudantes.

Esta padronização na linguagem foi importante, uma vez que o questionário a ser respondido pelos estudantes tratava de questões abertas. Com isso, aumentou-se a probabilidade de se obter respostas com palavras que identificassem elementos da ferramenta.

A demonstração da ferramenta foi feita através da apresentação de algumas funções construídas para exemplo. O objetivo dos exemplos foi demonstrar a ação de cada um dos elementos da interface (áreas, botões, etc) e, principalmente, os elementos das funções exibidos na tela, destacando-se as representações visuais das Estruturas de Dados.

Após a demonstração, e respondidas as questões levantadas pelos estudantes, passou-se para a aplicação do questionário. As questões elaboradas para o questionário foram abertas de forma a permitir que os estudantes pudessem discorrer livremente sobre a impressão que tiveram sobre a ferramenta.

Na elaboração das questões buscou-se não incluir palavras que pudessem induzir o estudante a utilizá-la como elemento em sua resposta. Por exemplo, um dos componentes importantes da ferramenta é a possibilidade da execução passo-a-passo, controlada pelo estudante. A expressão "passo-a-passo" não foi emprega na formulação das questões. Entretanto, 80% dos estudantes citaram a execução passo-a-passo como um recurso facilitador no processo de compreensão da demonstração da função.

7.4 – Análise da pesquisa

As seis questões apresentadas no questionário para os estudantes responderem foram:

1. Você acha que a ferramenta ajudará a entender o uso das estruturas de dados como tipo de dado abstrato?
2. Você acha que demonstrar o algoritmo ajudará a identificar o erro em uma função e consequentemente corrigi-lo?
3. O que você achou da interface?
4. As representações das estruturas de dados (lista linear, pilhas e filas) correspondem com as que foram dadas em sala de aula?
5. Como foi o seu desempenho na matéria de Orientação a Objetos? Qual foi seu grau de dificuldade?
6. Opinião geral sobre a ferramenta. Sugestões e Críticas.

Para as quatro primeiras questões, 100% dos estudantes responderam de forma positiva. Os alunos consideram: que a ferramenta ajudará a entender o uso de estruturas de dados como tipo de dado abstrato; que a demonstração do algoritmo ajuda a identificar o erro em uma função; que a interface é amigável; que

as representações das estruturas de dados correspondem às apresentadas em sala de aula.

A questão de número cinco tem a finalidade de buscar uma correlação entre o aprendizado das estruturas de dados como tipo de dado abstrato, e a facilidade de aprendizado na disciplina sobre orientação a objetos, que sucede a disciplina discutida neste trabalho. O desempenho "bom" foi relatado por 50% deles, e "regular" por 33,4% deles. Apenas um dos estudantes não havia feito a matéria sobre orientação a objetos.

Quanto ao grau de dificuldade encontrado na dificuldade sobre orientação a objetos, 20% declararam que tiveram um "alto" grau de dificuldade, 60% "médio" e 20% "baixo".

Para questão de número seis, que trata da opinião geral, foi deixado bem claro na apresentação que as críticas existentes deveriam ser relatadas pois representam também elementos importantes para a pesquisa.

Esta questão não foi respondida por apenas um dos estudantes e, apenas um deles, fez sugestões.

A seguir são apresentadas algumas observações colhidas do questionário que trazem elementos sobre as características principais da ADTTool:

..."a ferramenta será muito útil para os alunos do primeiro ano que, geralmente, têm muita dificuldade no começo"...

..."o aluno pode ver passo a passo o que está acontecendo com o algoritmo e as variáveis"...

..."não fará confusões sobre implementação já que o aluno não vai saber como foi feito"...

..."você tem algo concreto para observar"...

..."é de extrema importância ter um algoritmo para o entendimento do processo"...

..."a interface é simples e fácil de usar"...

..."as representações visuais são exatamente iguais às mostradas no quadro"...

CONCLUSÕES

O trabalho realizado e, em especial a ferramenta Abstract Data Type Tool - ADTTool, dentro de suas restrições em relação aos tipos de dados definidos, se propõe a cobrir um espaço vazio existente entre o desenvolvimento de um programa e sua execução final, possibilitando o execução de uma função isolada.

Adicionalmente, utilizando-se toda a potencialidade dos recursos visuais que causam interesse no uso das aplicações por parte dos estudantes e por trazer uma bagagem de facilitador do aprendizado, a ferramenta não apresenta simplesmente um resultado final da funcionalidade de seu conteúdo, mas faz a animação da função através da demonstração visual dos passos que envolvem o fluxo de dados.

Na proposta dos elementos que compõem a interface, procurou-se estabelecer representações intuitivas e significativas para cada uma das estruturas de dados, de maneira que a informação apresentada não se mostrasse textual, necessitando ser buscada dentro de um conjunto de informações.

Nas representações visuais criadas, apesar de simples, buscou-se manter a abstração na medida em que não se utilizaram elementos que, de alguma forma, vinculassem a estrutura de dado com algum tipo de representação interna, escondendo, assim, detalhes da representação.

Com o interpretador buscou-se manter o padrão da linguagem C++ uma vez que as funções a serem testadas são editadas em um ambiente de programação desta linguagem e que, após serem testadas, deverão ser reintegradas ao produto maior que é o programa. Além disso, deve atender o objetivo da criação da ferramenta que se coloca como um componente de execução no fluxo a ser seguido no desenvolvimento de um programa, se utilizado o ambiente da linguagem C++.

A escolha do ambiente de desenvolvimento da ferramenta baseou-se na gama de recursos gráficos e visuais existentes na biblioteca de classes MFC do ambiente Visual C++, propiciando a criação de uma interface atraente e com recursos de fácil reconhecimento por parte dos estudantes, tais como as janelas e os botões.

A possibilidade de manipular e controlar uma aplicação ao invés de situar-se apenas como espectador exerce uma influência muito grande nos usuários, de modo geral. Assim, na criação dos elementos de controle da animação do algoritmo, buscou-se estabelecer um recurso que pudesse oferecer a cada estudante o seu próprio ritmo para observar a animação de sua função. O recurso de executar "passo a passo", controlado pelo acionamento de uma tecla para que o passo seguinte seja executado e não simplesmente determinando um tempo entre uma demonstração e outra, é que torna possível ao estudante analisar e verificar a existência, ou não, de erros em sua função, combinado com o tempo que cada um precisa para isto.

Contribuições

Este trabalho apresenta como principal contribuição a ferramenta propriamente dita, que oferece um ambiente de execução para funções isoladas, criadas pelos alunos, utilizando como abordagem principal as estruturas de dados como tipo de dado abstrato. Como contribuição complementar, a ferramenta possui uma interface gráfica que faz a animação da função demonstrando o fluxo de dados entre as estruturas.

Apresenta ainda contribuições complementares que auxiliaram na composição deste trabalho:

- uma proposta de formalismo na definição do tipo de dado abstrato com a inclusão da representação visual abstrata;
- uma relação, com classificação e comentários, de algumas ferramentas que fazem a animação ou demonstração de algum conceito ligado a ciência da computação.

Melhorias para Próxima Versão

A ferramenta ADTTool está completa e operacional, apesar das limitações em relação ao tipo da informação que se quer armazenar nas estruturas de dados. Como sugestões para as próxima versão destacam-se:

- Introdução de recursos para a entrada de dados pelo aluno, e não a geração automática como na versão atual;
- Implantação de recursos que possibilitem voltar ao passo anterior utilizando-se, por exemplo, a tecla “seta esquerda”;
- Incluir a edição dentro da Área de Listagem do programa com o objetivo de corrigir algum erro à medida que o estudante o detecta;
- Completar o interpretador para os outros recursos da linguagem que não foram implementados.

Continuidade da Pesquisa

A continuidade deste trabalho converge para duas situações importantes: as aplicações decorrentes da ferramenta ADTTool propriamente dita e as pesquisa futuras.

A primeira situação, ligada diretamente à ferramenta ADTTool, pode destacar dois pontos: a migração para uma tecnologia para sua utilização na *web* e a sua expansão para ser utilizada também como auxiliar no ensino de programação na linguagem C. Uma situação complementar a esta, porém ainda ligada à ferramenta, trata da troca do interpretador de uma função escrita em C para um interpretador de uma linguagem algorítmica de tal forma que possa ser aplicada em outros ambientes de ensino.

A segunda situação, mais ampla, é a que utiliza a experiência deste trabalho como base de pesquisa para a criação de um laboratório de tecnologias de software, envolvendo metodologias de ensino, para a criação de ferramentas para auxiliar no ensino em outras áreas de conhecimento. Esta base de pesquisa está vinculada ao grupo de pesquisa ligado à tecnologias de apoio ao ensino da PUC-Campinas.

Bibliografia

[AMB97] Ambler, Allen L.; Green, Thomas; Kumura, Takayuki Da; Repenning, Alexander Smedley – "1997 Visual Programming Challenge Summary" – Proceedings of 1997 IEEE Symposium on Visual Languages – Sept 23-26, 1997 – Isle of Capri, Italy.

[ANT00] Antony, Sergio; Hamlet, Dick; "Automatically Checking an Implementation against Its Formal Specification" - IEEE Transaction Software Engineering - V26 n°1 – 2000.

[AST98] Astrachan, Owen; Rodger, Susan H. – "Animation, Visualization, and Interaction in CS1 Assignments" – Proceedings on the 29th SIGCSE Technical Symposium on Computer Science Education, Atlanta GA – USA – 1998.

[BAK99] Baker, Ryan; Boilen, Michael; Goodrich, Michael T; Tamassia, Roberto; Stibel, B. Aaron – "Testers and Visualizers for Teaching Data Structures" – Proceedings on the 30th SIGCSE Technical Symposium on Computer Science Education , New Orleans, LA – USA – 1999.

[BER75] Bertiss, A.T. – "Data Structures – Theory and Practice " – 2nd Edition – Academic Press, Inc – 1975.

[BOR96] Boroni, Christopher M.; Eneboe, Torlief J.; Goosey, Frances W.; et al. – "Dancing with DynaLab" – Proceedings on the 27th SIGCSE Technical Symposium on Computer Science Education – Feb/96, Philadelphia, PA – 1996.

[BOR97] Boroni, C; Goosey, F; Grinder, M; Rockford, R; Wissemback, P - "WebLab! A Universal and Interactive Teaching, Learning and Laboratories Environment for the Word Wide Web." - Proceedings on the 28th SIGCSE Technical Symposium on Computer Science Education - (1:199-203 - March) – 1997.

[BRI00] Bridgeman, S.; Goodrich, M. T.; Kobourov, S. G.; Tamassia, R. – "PILOT: An Interactive Tool for Learning and Grading" - Proceedings on the 31st SIGCSE Technical Symposium on Computer Science Education – Austin, TX – USA – 2000.

[BRO84] Brown, Marc H.; Sedgewick, Robert – "A System for Algorithm Animator" – ACM Computer Graphics, Vol. 18, N° 3 – 1984.

[BRO88] Brown, Marc H. – "Exploring Algorithms BalsaII" – IEEE Computer - Vol. 21, Nº 5 – May 1988.

[BRO92] Brown, Marc H. – "An Introduction to Zeus: Audiovisualization of Some Elementary Sequential and Parallel Sorting Algorithms" – Proceedings of the Conference on Human Factors and Computing System - CHI'92 – May 3-7, 1992.

[BRO97] Brown, Marc H.; Najorh, Marc A.; Raisanmo, Roope – "A Java-Based Implementation of Collaborative Active Textbooks" – Proceedings of the 1997 IEEE Symposium on Visual Languages – Sept 23-26, 1997 – Isle of Capri, Italy.

[BRU97] Brummund, Peter - "Educational Animations" – Hope College REU – http://www.cs.hope.edu/~alganin/ccaa/ccaa.html – 1997.

[BUC01] Bucci, Paolo; Long, Timothy J.; Weide, Bruce W. – "Do We Really Teach Abstraction?" – Proceedings on the 32nd SIGCSE Technical Symposium on Computer Science Education, Charlotte, NC - USA – 2001.

[BUR98] Burnet, Margaret M.; Gottfried, Herkimer J. – "Graphical Definitions: Expanding Spreadsheet Languages through Direct Manipulation and Gestures" – ACM Transactions on Computer-Human Interaction, Vol. 5, N°. 1, March 1998.

[CAL94] Calloni, Ben A.; Bagert, Donald J. – "ICONIC Programming in BACCII © Vs Textual Programming: Which is a Better Learning Environment?" - Proceedings of the 25th SIGCSE Technical Symposium on Computer Science Education – Phoenix, Arizona, USA – 1994.

[CAL95] Calloni, Ben A; Bagert, Donald J. – "Iconic Programming for Teaching the first Year Programming Sequence" – Frontiers in Education Conference – 1995.

[CAL97] Calloni, Ben A.; Bagert, Donald J. – "Iconic Programming Proves Effective for Teaching the First Year Programming Sequence " - Proceedings of the 28th SIGCSE Technical Symposium on Computer Science Education – CA, USA – 1997.

[CAP98] Capobianco, F.; Mosconi, M. – "Visual Programming in a Distributed Environment" - Proceedings of the 1998 IEEE Symposium on Visual Languages – Sept 1-4 – Halifax, Nova Scotia, Canada – 1998.

[CHA97] Lam, Stephen W. C.; Chan, Keith C. C.; Leung, Hareton K. N. ;Chung, Lawrence M.L. – "A Visual Programming Environment for Z Specification" – Proceedings of the 4th Asia-Pacific Software Engineering and International Computer Science Conference (APSEC'97/ICSC'97) - 1997.

[CHA00] Champoux, Bernard; David, Martin; Huot, Alain - "CAILS: A Prototype for a Computer Assisted Iconic Language System" – International IEEE Symposium on Visual Languages (VL'00)- Seattle, Washington – 2000.

[CON99] Concepcion, Arturo I, Cummins, Lawrence E, Moran, Ernest J, Do, Man M. – "Algorithma98: An Algorithm Animation Project" - Proceedings of the 30th SIGCSE Technical Symposium on Computer Science Education - New Orleans, LA – USA – 1999.

[COS97] Costagliola, G.; De Lucia, A.; Orefice, S.;Tortora, G. – "A Framework of Syntatic Models for the Implementation of Visual Languages" – Proceedings of the 1997 IEEE Symposium on Visual Languages – Sept 23-26, 1997 – Isle of Capri, Italy.

[COX85] Cox, Philip T.; Mulligan, I.J. – "Compiling the Graphical Functional Language PROGRAPH" – Proceedings of the 1985 ACM SIGSMALL Symposium on Small Systems – May 1985 – Danvers, Massachusetts, USA.

[COX94] Cox, Philip T.; Smedley, Trevor J. – "Using Visual Programming to Extend the Power of Spreadsheet Computation" – Proceedings of the Workshop on Advanced Visual Interfaces – AVI'94 – 1994 – Bari, Italy

[DER98] Dershem, Herbert L.; Vanderrhyde, James – "Java Class Visualization for Teaching Object-Oriented Concepts" - Proceedings of the 29th SIGCSE Technical Symposium on Computer Science Education, Atlanta, GA – USA – 1998.

[DER99] Dershem, Herbert L.; Parker, Daisy Erin; Weinhold, Rebecca - "A Java Function Visualizer" – Journal of Computing in Small Colleges, 15,1 – Nov. 1999 – pp. 222-230 - Hope College.

[DJA98] Djang, Rebecca W.; Burnett, Margaret M. – "Similarity Inheritance: A New Model of Inheritance for Spreadsheet VPLs" - Proceedings of the 1998 IEEE Symposium on Visual Languages – Sept 1-4 – Halifax, Nova Scotia, Canada – 1998.

[DOM97] Domingue, John; Mulholland, Paul – "Staging Software Visualizations on the Web" – Proceedings of the 1997 IEEE Symposium on Visual Languages – Sept 23-26, 1997 – Isle of Capri, Italy.

[ENG01] Engelbrecht, A.M. – "Implementação da Ferramenta ADTTool" – Relatório Técnico - 2001.

[ENG02] Engelbrecht, A.M. – "Implementação do Algoritmo para Enumeração de Todos os Caminhos sem Repetições do Dígrafo que Representa os Botões da Interface da ADTTool" – Relatório Técnico - 2002.

[ERW00] Erwig, Martin - "A Visual Language for XML" - International IEEE Symposium on Visual Languages (VL'00) – Seattle, Washington – USA – 2000.

[GEI98] Geiger, Christian; Mueller, Wolfgang; Rosenbach, Waldemar – "SAM – An Animated 3D Programming Language" – Proceedings of the 1998 IEEE Symposium on Visual Languages – Sept 1-4 – Halifax, Nova Scotia, Canada – 1998.

[GEO00] George, Carlisle E. – "EROSI - Visualising Recursion and Discovering New Errors" – Proceedings of the 31st SIGCSE Technical Symposium on Computer Science Education - Austin, TX – USA – 2000.

[GHI98] Ghittori, Elena; Mosconi, Mauro; Porta, Marco – "Designing New Programming Constructs in a Data Flow VL" – Proceedings of the 1998 IEEE Symposium on Visual Languages – Sept 1-4 – Halifax, Nova Scotia, Canada – 1998.

[GOL97] Goldberg, Murray W. – "CALOS:First Results From an Experiment in Computer-Aided Learning for Operating Systems" - Proceedings of the 28th SIGCSE Technical Symposium on Computer Science Education – CA, USA – 1997.

[GUR96] Gurka, Judith S.; Citrin, Wayne - "Testing Effectiveness of Algorithm Animation" – Proceedings of the 1996 IEEE Symposium on Visual Languages (VI'96) – Boulder, Colorado – USA – 1996.

[HAA97] Haajanen, J.; Pesonius, M.; Sutinen, E.; et al. – "Animation of User Algorithms on the Web" - Proceedings of the 1997 IEEE Symposium on Visual Languages – Sept 23-26, 1997 – Isle of Capri, Italy.

[HAR94] Hartley, Stephen J. – "Animating Operating Systems Algorithms with XTANGO" – Proceedings of the 25th SIGCSE Technical Symposium on Computer Science Education - Phoenix, Arizona, USA – 1994.

[HEG97] Hegazi, Ahmed F.; Metwally, Ahmed M.; Degady, Lamia M.; et al – "Visual Craft: A Visual Integrated Development Environment" – Proceedings of the 2nd IEEE Symposium on Computers and Communications – 1997.

[HOF00] Hoffmann, Berthold; Minas, Mark - "Towards Generic Rule-Based Visual Programming" - Proceedings of the 2000 IEEE International Symposium on Visual Languages- Sept. 10-13, 2000 – Seattle, Washington – USA.

[HOR87] Horowitz, Ellis; Sahni, Sartaj – "Fundamentos de Estruturas de Dados" – Editora Campus – 3ª Edição – 1987.

[HUM96] Humphrey, Watts S. – "Using a Defined and Measured Personal Software Process" – IEEE Software – Vol. 13, Nº 3 – May 1996.

[HUT86] Hutchins, Edwin L.; Hollan, James D. e Norman, Donald A. - "Direct Manipulation Interfaces" – capítulo do livro: "User Centered System Design" – Editado por Donald A. Norman e Stephen W. Draper – Lawrence Erlbaum Associates, Publishers – 1986.

[IBR98] Ibrahim, Bertrand – "Diagrammatic Representation of Data Types and Data Manipulations in a Combined Data- and Control Flow" – Proceedings of the 1998 IEEE Symposium on Visual Languages – Sept 1-4 – Halifax, Nova Scotia, Canada – 1998.

[JAC97] Jackson, David; Usher, Michelle – "Grading Student Programs using ASSYST" – Proceedings of the 28th SIGCSE Technical Symposium on Computer Science Education – CA, USA – pages. 335 – 339.

[JAR00] Jarc, Duane J.; Feldman, Michael B.; Heller Rachelle S. – "Assessing the Benefits of Interactive Prediction Using Web-based Algorithm Animation Courseware" – Proceedings of the 31st SIGCSE Technical Symposium on Computer Science Education – Austin, TX.

[JAR98] Jarc, Duane J., Feldman, Michael B. – "An Empirical Study of Web-based Algorithm Animation Courseware in a Ada Data Structure Course" – Proceedings of the Annual ACM SIGAda International Conference on Ada - Washington, D.C. – USA – 1998.

[JUM97] Al-Jumeily, D.; Strickland, P.- "Designing an interface on the web for an Intelligent Tutoring System" - 23rd EUROMICRO Conference'97, New Frontiers of Information Technology – 1997.

[KAP00] Kaplan, Alan, Shoup, Denise – "CUPV: A Visualization Tool for Generated Parsers" – Proceedings of the 31st SIGCSE Technical Symposium on Computer Science Education - Austin, TX – USA – 2000.

[KRA00] Krames, Michael D.; Zhang, Du - "GAPS: a Generic Programming System" – Proceedings of the Twenty-Fourth International Computer Software & Applications Conference (COMPSAC) – 2000.

[LAH96] Lahtinen, S.-P; Lamminjoki, T.; Sutinen, E.; Tarhio, J.; Tuovinen, A.-P. – "Towards Automated Animation of Algorithms" – Proceedings of the Fourth International Conference in Central Europe on Computer Graphics and Visualization – 1996.

[LAN96] Langsam, Y.; Augenstein, M.J.; Tenenbaum, A.M. – "Data Structures Using C and C++" – 2nd Edition – Prentice – Hall Inc – 1996.

[LI 97] LI, Xiaosong; Mugridge, Warwick B.; Hosking, John G. – "A Petri Net-Based Visual Language for Specifying GUIs" – IEEE Symposium on Visual Languages, VL'97 – Sept. 23-26 – 1997 – Isle of Capri, Italy.

[LIM99] Lima, Lauro de Oliveira – "Piaget: Sugestões aos Educadores"- Editora Vozes – 1999.

[LIS74] Liskov, Barbara; Zilles, Stephen – "Programming with Abstract Data Types" – ACM SIGPLAN Notices, 9(4):50-59, 1974.

[MCW96] McWirter, Jeffrey D. – "AlgorithmExplorer: A Student Centered Algorithm Animation System" - Proceedings of the 1996 IEEE Symposium on Visual Languages (VI'96) – Boulder, Colorado – USA – 1996.

[MIC96] Michail, Amir - "Teaching Binary Tree Algorithms through Visual Programming" - Proceedings of the 1996 IEEE Symposium on Visual Languages (VI'96) - Boulder, Colorado – USA – 1996.

[MÜN98] Münch, M; Schürr, A.; Winter, A. – "Integrity Constraints in the multi-paradigm language PROGRES" – Proceedings of the IEEE Symposium on Visual Languages – Sept. 1-4, 1998 – Halifax, Nova Scotia, Canada.

[MYE00] Myers, Brad; Hudson, Scott E.; Pausch, Randy - "Past, Present, and Future of User Interface Software Tools" - ACM Transactions on Computer-Human Interaction, Vol 7, No. 1, March 2000.

[MYE97] Myres, Brad A.; McDaniel, Richar G.; Miller, Robert C.; et al. – "The Amulet Environment: New Models for Effective User Interface Software Development" - IEEE Transactions on Software Engineering, Vol. 23, Nº. 6, June 1997.

[NAJ01] Najork, Marc A. – "Web-based Algorithm Animation" – Proceedings of the 38th conference on Design Automation- ACM, June 18-22, Las Vegas, Nevada – 2001.

[NAP90] Naps, Thomas – "GAIGS – User's Manual (Version 3.0 for PC)" – Lawrence University- Appleton, WI – 1990.

[NAP96] Naps, Thomas L., Stenglein, Jeremy - "Tools for Visual Exploration of Scope and Parameter Passing in a Programming Languages Course" - Proceedings of the 27 th SIGCSE Technical Symposium on Computer Science Education – Philadelphia, PA – USA – 1996.

[NAR98] Narayanan, N. Hari; Hübscher, Roland – "Visual Language Theory: Towards a Human-Computer Interaction Perspective" – Capítulo 3, do Livro: "Visual Language Theory" - Springer - Editors: Kim Marriot and Bernd Meyer –1998.

[NIE93] Nielsen, Jakob – "Usability Engineering" – Morgan Kaufmann Publishers, Inc. – 1993.

[NOR94] Norman, Donald A. – "Things That Make Us Smart" – Perseus Books – 1994.

[PFE00] Pfeiffer Jr, Joseph J.; Vinyard Jr, Rick L.; Margolis, Bernardo - "A Common Framework for Input, Processing, and Output in a Rule-based Visual Language" – Proceedings of the 2000 IEEE International Symposium on Visual Languages(VL'00) – Seattle, Washington – USA – 2000.

[PFE97] Pfeiffer Jr., Joseph J. – "A Rule-Based Visual Language for Small Mobile Robots" - Proceedings of the 1997 IEEE Symposium on Visual Languages – Sept 23-26, 1997 – Isle of Capri, Italy.

[PFE99] Pfeiffer, Joseph Jr.; "A Language for Geometric Reasoning in Mobile Robots" – Proceedings of the IEEE Symposium on Visual Languages– – Tokyo, Japan - Sept. 1999.

[PIE98] Pierson, Willard C.; Rodger, Susan H. – "Web-based Animation of Data Structures Using JAWAA" - Proceedings of the 29th SIGCSE Technical Symposium on Computer Science Education - Atlanta GA – USA – 1998.

[PIL00] Pilgrim, Matthew; Bouchlaghem, Dino; Loveday, Dennis; et al – "Abstract Data Visualisation in the Built Environment" - IEEE International Conference on Information Visualization – 2000.

[PRE01] Prechelt, Lutz - "Accelerating Learning from Experince: Avoiding Defects Faster" - IEEE Software –Vol. 18, Nº6 – November/December - 2001.

[PRE92] Pressman, Roger S. – "Software Engineering: a Practitioner's Approach" – McGraw-Hill International Editions – Third Edition – 1992.

[RAS01] Rasala, Richard; Raab, Jeff; Proulx, Viera K. – "Java Power Tools: Model Software for Teaching Object-Orientd Design" - Proceedings of the 32nd SIGCSE Technical Symposium on Computer Science Education – Charlotte, NC USA – 2001.

[ROB01] RöBling, Guido; Freisleben, Bernd – " AnimalScript: An Extensible Scripting Language for Algorithm Animation" - Proceedings of the 32nd SIGCSE Technical Symposium on Computer Science Education Charlotte, NC USA – 2001.

[ROB91] Robson, Robert; Seminar, Kudang – "Visual Editing of Data Structures" – Proceedings of the IEEE Conference on Software – 1991.

[SCH00] Schär, Sissel Guttormsen; Kruger, Helmut - "Using New Learning Technologies with Multimedia" - IEEE Multimedia – Vol. 7, Nº 3 - July-September 2000.

[SCH95] Schürr, A.; Winter, A.; Zündorf,A. – "Visual Programming with Graph Rewriting Systems" – Proceedings of the 11th International IEEE Symposium on Visual Languages – Sept. 05-09, Darmstadt, Germany- pages 326-333 – 1995.

[SCH97] Schmidt, Bobby – "The Learning C/C++ - Driving You to Abstraction" - C/C++ Users Journal 15,1 (Jan. 1997), Article 8 – 1997.

[SEB98] Sebillo, Monica; Tortora, Genoveffa; Vitiello, Giuliana – "An Iconic Environment for the Definition of Visual DBMSs" – Proceedings of the 1998 IEEE Symposium on Visual Languages – Sept 1-4 – Halifax, Nova Scotia, Canada – 1998.

[SEN94] Sengupta, Saumyendra; Korobkin, Carl P. – "C++ , Object-Oriented Data Structures" – Springer-Verlag New York, Inc. – 1994.

[SHN98] Shneiderman, Ben – "Designing the User Interface" – Addison Weley –1998.

[SHN00] Shneiderman, Ben - "Universal Usability" – Communications of the ACM - Vol. 43, No. 5 - May 2000.

[SMI94] Smith, David C. – "KIDSIM: Programming Agents Without a Programming Language" – Communications of the ACM – Vol. 37, N°. 7 - July 1994.

[STA90] Stasko, John T - "Tango: A Framework and System for Algorithm Animation" – IEEE Computer 23,9 - September 1990 – 1990.

[STA94] Stasko, J – "POLKA Animation Designer's Package – Animator's Manual" – included in Polka Software Documentation – 1994.

[STA01] Stasko,J – "XTANGO" In. http://www.cc.gatech.edu .

[TOW78] Towle, Tom; DeFanti, Tom – "Gain: An Interactive Computer Graphics Programming" – Proceedings of the 5th Annual Conference on Computer Graphics and Interactive Techniques –SIGGRAPH'78 – 1978.

[TUR01] Turner, Joseph A.; Zachary, Joseph L. – "Javiva: A Tool for Visualizing and Validating Student-Written Java Programs" – Proceedings of the 32nd SIGCSE Technical Symposium on Computer Science Education – Charlotte, NC USA – 2001.

[VOD95] Vodislav, Dan - "Visual Programming for Animation in User Interface" - Proceedings of the 11th International IEEE Symposium on Visual Languages (VL'95) – 1995.

[VOD97] Vodislav, Dan – “A Visual Programming Model for User Interface Animation” – Proceedings of the 1997 IEEE Symposium on Visual Languages – Sept 23-26 – Isle of Capri, Italy- 1997.

[WOD97] Wodhi, R.; Cull, P.- “Calypso: A Visual Language for Data Structures Programming” - Proceedings of the 1997 IEEE Symposium on Visual Languages – Sept 23-26, 1997 – Isle of Capri, Italy – 1997.

ANEXO I

Introdução

Neste anexo são apresentadas ferramentas que foram analisadas, no decorrer do desenvolvimento do trabalho. Estas ferramentas estão categorizadas de acordo com características que possuem e que foram objeto de comparação com a ADTTool. As características referentes às categorias 1, 2 e 3 foram abordadas no Capítulo II. As demais categorias não possuem características que as relacionam com ADTTool, porém foram analisadas por se tratarem de ferramentas de animação e de ambientes que possuem interfaces gráficas. As categorias são:

1. Ambientes para a criação de animações através da inclusão de diretivas nos algoritmos;
2. Ferramentas que demonstram de alguma forma as Estruturas de Dados: pilha, filas, listas lineares, árvores e grafos;
3. Ferramentas de animação de Métodos de Conhecimento Geral, tais como os métodos de classificação e recuperação da informação. Estas ferramentas demonstram também operações que manipulam Estruturas de Dados;
4. Ferramentas que fazem a animação de elementos dos algoritmos tais como: passagem de parâmetros, seqüência de chamadas das funções, recursividade;
5. Ferramentas que fazem a animação ou demonstração, de formas variadas, de classes em algoritmos construídos com o paradigma de orientação à objetos;

6. Ferramentas com Interfaces Gráficas de linguagens que fazem a programação ou manipulação de robôs;
7. Ferramentas para a criação de programas utilizando ambientes com interfaces gráficas;
8. Ferramentas que permitem a interação do estudante com o professor para correção e solução de programas;
9. Outros ambientes com Interfaces Gráficas para: especificação de programas, sistema para programação genética, processamento de linguagem natural, etc.

1. Ambientes para a criação de animações através da inclusão de diretivas nos algoritmos

GAIGS **1988 [NAP90]**

É um sistema gráfico para criar visualizações em programas. A imagem gráfica de estruturas de dados é produzida a partir de informações contidas num arquivo texto, em formato especial, criado pelo usuário. Trabalha com as estruturas: pilha, fla, lista ligada, arranjos de uma e duas dimensões, grafos, redes, árvores binárias e árvores gerais. É uma aplicação para Windows, escrita em Pascal. A primeira versão foi desenvolvida em 1988. O manual do usuário pode ser encontrado no endereço: http://www.lawrence.edu/dept/compsci/research.html.

TANGO (Transition based Animation GeneratiOn) **1989 [STA90]**

A descrição desta ferramenta encontra-se no Capítulo II. Uma cópia das ferramentas TANGO e XTANGO pode ser obtida livremente no endereço: http://www.cc.gatech.edu/gvu;

ZEUS **1991 [BRO92]**

Zeus é um sistema de animação de algoritmo, sucessor do sistema BALSA [BRO88], descrito no Capítulo II. Zeus possui mais recursos, tais como: cor, som, paralelismo e a entrada de dados pelo usuário. Foi escrito em Modula-2. No endereço: http://www.research.compaq.com/SRC/zeus/home.html, existem vários exemplos de animações construídos com o ZEUS;

POLKA **1993 [STA94]**

POLKA é um ambiente para a construção de animação de programas e é o sucessor de XTANGO [STA01], descrito no Capítulo II. É considerado mais poderoso e flexível, pois suporta animação de computação e programação paralela, além de trabalhar com cores e aplicações em tempo real.
Originalmente foi escrito em C++ sobre Unix e Xwindows. POLKAW é a versão para Windows. ftp.cc.gatech.edu/pub/people/stasko;

AlgorithmExplorer **1996 [MCW96]**

É um sistema para animação de algoritmos que permite aos estudantes desenvolver e interagir com animação de programas criados por eles. Permite a entrada de dados pelo usuário e possui uma interface com comandos de animação.

Possui três componentes funcionais: escolha de tipos de dados, linguagem declarativa de comandos de animação e linguagem operacional de comandos de animação. No primeiro componente são escolhidos, em uma biblioteca, os tipos de dados que o estudante deseja animar. A linguagem declarativa consiste de declarações simples que permitem manipular e criar os objetos gráficos fornecidos pelo sistema. O último componente faz a interpretação das estruturas de dados e as declarações criadas. A linguagem declarativa foi inspirada no sistema XTANGO[STA01].

JCAT (Java – Collaborative Active Textbooks) 1997 [NAJ01] e [BRO97]

É um sistema para a criação de animação em algoritmos escritos em Java. É a versão Java do sistema CAT, criado em 1995. A estratégia para animar os algoritmos segue o mesmo modelo de BALSA [BRO84], onde são anotados, dentro do programa, pontos de interesse que interpretados pela ferramenta, geram animações. No endereço http://www.research.compaq.com/SRC/JCAT/pode-se encontrar mais informações sobre a ferramenta. A Figura I.1 apresenta uma tela de exemplo da ferramenta.

JAWAA (Java And Web-based Algorithm Animation) 1998 [PIE98]

A descrição desta ferramenta encontra-se no Capítulo II. Mais informações e alguns exemplos podem ser encontrados em: http://www.cs.duke.edu/csed/jawaa/JAWAA.html. A Figura I.2 mostra uma tela com um exemplo relacionado com filas.

AnimalScript 2000 [ROB01]

É uma linguagem para criação de animação. Consiste da entrada de um arquivo com linhas de comando em ASCII, onde cada linha pode conter um comentário ou um comando escrito em AnimalScript, que é composta de primitivas gráficas. Os comandos básicos possuem os efeitos de animação: mostre, esconda, mova, rotacione, mude a cor e comandos para desenhar polígonos e linhas. Possui um interpretador dos comandos. No endereço www.informatik.uni-siegen.de/~roesslin/Animal/index.html, pode-se encontrar exemplos do uso da ferramenta. A Figura I.3 mostra o exemplo de uma aplicação.

2. Ferramentas que demonstram de alguma forma as Estruturas de Dados: pilha, filas, listas lineares, árvores e grafos

DIVA 1998 [IBR98]

É um ambiente de desenvolvimento de software baseado em uma linguagem de especificação semi – formal e no fluxo de dados e de

controle. Foi desenvolvido a partir de um sistema chamado IDEAL, direcionado para o desenvolvimento de material de aprendizado apoiado por computador que somente suportava formalismo de fluxo de controle. O ambiente possui vários editores com recursos que combinam o fluxo de dados e o fluxo de controle, recursos para a descrição de tipos de dados e a instanciação de estruturas de dados. Para conciliar num mesmo diagrama o fluxo de dados e o de controle, foi utilizado um grafo para a representação onde os nós, que representam atividades, são conectados através de arcos por onde fluem os dados e os componentes de controle. Os componentes de controle não possuem representação interna e atravessam o grafo ao longo dos arcos do fluxo de controle. Os componentes de dados são definidos através de tipos de dados e atravessam o grafo pelos arcos do fluxo de dados. Entre os tipos de dados representados estão os tipos de dados estáticos: inteiros, reais, lógicos, intervalos de inteiros, caracteres e cadeias de caracteres, e entre os tipos de dados estruturados estão os arranjos e os registros. As listas são representadas através de arranjos.

DynaLab (Dynamic Laboratory) 1995 [BOR96]

É um sistema interativo criado como elemento motivador para as aulas de laboratórios em ciência da computação, que utiliza recursos visuais. Faz a animação de algoritmos como os métodos de ordenação e animação de conceitos (por exemplo máquinas de *Turing*), ao mesmo tempo em que interage com o aluno efetuando avaliações de aprendizado. A demonstração de estruturas de dados é feita através de elementos textuais onde é mostrado o nome da variável e o valor que recebe. A versão completa de 1995 previa apenas algoritmos em Pascal, em [BOR97] foram apresentadas considerações sobre a migração do DynaLab para a web. No endereço http://www.cs.montana.edu/~dynalab, pode-se encontrar outras informações sobre o DynalLab. A Figura I.4 apresenta um exemplo do uso desta ferramenta.

Opsis **1996** **[MIC96]**

É uma applet em Java que auxilia no ensino de árvores binárias balanceadas. A representação gráfica da árvore é feita através de triângulos, representando expansões da árvore, conectados aos respectivos nós pai. No endereço http://www.cs.washington.edu/homes/Opsis.html, pode-se executar a applet. A Figura I.5 mostra um exemplo do uso da ferramenta.

JELIOT **1997** **[HAA97]**

Esta ferramenta está descrita no Capítulo II. Outras informações e exemplos de aplicações podem ser encontradas no endereço: http://www.cs.helsinki.fi/research/aaps/Jeliot. A Figura I.6 apresenta um exemplo usando uma fila.

JDSL (Data Structures Library in Java;) **1999** **[BAK99]**

É composto de dois componentes: JDSL Visualizer e o JDSL Testers. O primeiro é uma ferramenta para visualizar as estruturas de dados e para isto possui uma biblioteca de interfaces (API's) para cada tipo e essas interfaces devem ser invocadas dentro do programa criado pelo estudante. As interfaces mostram os dados das estruturas no momento anterior e posterior à execução de um método referente à estrutura de dados. É possível demonstrar em qualquer momento do algoritmo a situação das estruturas, adicionando-se chamadas das interfaces. O segundo componente inclui o interpretador das funções, um recurso para manter um histórico dos passos executados. No endereço http://www.cs.brown.edu/cgc/jdsl pode-se obter uma cópia da ferramenta. A Figura I.7 mostra um exemplo de uma tela da ferramenta.

3. Ferramentas de animação de Métodos Clássicos, tais como os métodos de classificação e recuperação da informação. Estas ferramentas demonstram também operações que manipulam Estruturas de Dados

Animator 1997 [BRU97]

É uma ferramenta que faz a animação de algoritmos de ordenação. Além de apresentar a animação do método através de barras em movimento, apresenta o algoritmo com uma faixa colorida sobre a linha de comando que está sendo executada no momento. Escrito em Java e é executado na *web* e encontra-se no endereço http://www.cs.hope.edu/~alganim/animator/Animator.html. A Figura I.8 apresenta uma tela de um exemplo da ferramenta.

IDSV (Interactive Data Structure Visualization) 1998 [JAR98] e [JAR00]

Esta ferramenta está descrita no Capítulo II. O uso da ferramenta está disponível no endereço : http://www.student.seas.gwu.edu/~idsv/. A Figura I.9 apresenta uma tela exemplo desta ferramenta.

Algorithma98 1998 [CON99]

Esta ferramenta está descrita no Capítulo II. No endereço http://web.csusb.edu/public/class/cs455_1, além da versão do ano de 1998, encontram-se outras versões atualizadas. A Figura I.10 apresenta um exemplo de tela desta ferramenta.

4. Ferramentas que fazem a animação de elementos dos algoritmos tais como: passagem de parâmetros, seqüência de chamadas das funções, recursividade e outros

PSVE (Parameter and Scope Visualization Environment) 1996 [NAP96]

Ferramenta que explora a passagem de parâmetros e escopo de variáveis, através de representações visuais. Trata os vários tipos de passagem de parâmetros, tais como: por valor, por referência. Usa o sistema GAIGS [NAP90 e BRO97] de visualização para a demonstração visual de sua proposta. Escrito em Delphi para ambiente Windows.

Function Visualizer 1997 [DER99]

Ferramenta para visualização de funções escritas em Java com as seqüências de chamadas. A demonstração é feita em uma janela dividida ao meio. No lado direito apresenta o algoritmo com uma faixa colorida sobre o comando que está sendo demonstrado no momento. No lado esquerdo são apresentadas as variáveis e os valores que assumem em cada nova atribuição. Em funções recursivas, para cada chamada é aberta uma nova janela com o algoritmo e os novos valores dos parâmetros.

EROSI (Explicit Representer Of Subprogram Invocation) 2000 [GEO00]

Trata-se de um tutor visual para auxiliar na compreensão e uso de funções recursivas na solução de problemas. Utiliza a visualização e animação com som e cor para simular o modelo lógico dinâmico das chamadas recursivas de um subprograma. As funções são escritas em Pascal;

5. Ferramentas que fazem a animação ou demonstração, de formas variadas, de classes em algoritmos construídos com o paradigma de orientação à objetos

Object Visualizer **1998 [DER98]**

É um ambiente visual para auxiliar na compreensão de conceitos de orientação à objetos e facilitar testes e depurações. Permite a visualização de qualquer classe definida em Java. Uma classe é recebida como entrada e como resultado é produzida uma janela com a classe e seus métodos. Os objetos podem ser instanciados e manipulados dentro do ambiente.

Javiva **2001 [TUR01]**

Sistema para visualização de classes e respectivos métodos escritos em Java. Extrai pré-condições e pós-condições incluídos como comentários estilizados no programa fonte escrito em Java. Cria arquivos com as classes existentes e quando são executados, automaticamente, a ferramenta faz um diagnóstico e relata violações dos métodos definidos pelas pré e pós-condições. A animação das classes criadas é feita utilizando-se o sistema de visualização JDSL[BAK99].

6. Ferramentas com Interfaces Gráficas de linguagens que fazem a programação ou manipulação de robôs

KidSim (Kids' Simulations) **1994 [SMI94]**

Também conhecido como Cocoa. É uma ferramenta para crianças que permite que elas criem simulação simbólicas. É formado por um tabuleiro de jogo, dividido em espaços discretos como num tabuleiro de xadrez, um relógio, objetos de simulação, editor de regras e outros elementos. Possui uma coleção de objetos que são escolhidos e copiados para a cena que está sendo construída. As regras são utilizadas para definir o

comportamento dos personagens em cada cena. Essa ferramenta foi empregada para construir elementos de controle de em robô[AMB97] e [PFE00].

Altaira **1997** **[PFE97]**

É uma linguagem visual baseada em regras para pequenos robôs móveis. Combina informações de sensores, navegação e estado corrente, para determinar ações a serem tomadas pelo robô e as mudanças requeridas no estado. Possui um editor de definição do robô para permitir ao usuário especificar os sensores do robô e a configuração de atuação; um interpretador do conjunto de regras que permitirão ao robô operar sem ser preso à um hospedeiro e um mecanismo para intercalar o conjunto de regras análogas. O editor possui uma interface gráfica que permite ao usuário definir regras de forma completamente gráfica. Foram definidos ícones para entradas de direção e sensores, para o motor e saídas de navegação. É implementado em C++ e é o predecessor de Isaac [PFE00].

Isaac **1999** **[PFE99]**

Linguagem visual baseada em regras para aplicação em controle de robôs. Consiste de quatro editores integrados cuja manipulação é através de uma interface gráfica, um editor de objetos para a definição das bibliotecas, editor de entradas e saídas, editor para instanciar os sensores do robô, e um editor de regras.

7. Ferramentas para a criação de programas utilizando ambientes com interfaces gráficas

Prograph **1984** **[COX85] e [COX94]**

É uma linguagem de programação visual baseada no modelo de fluxo de dados e produz uma representação visual da hierarquia das classes e das definições. Em Prograph os elementos do programa são representados por diagramas de fluxo de dados. Possui uma lista de tipos de dados primitivos.

A primeira versão é de 1984. A Figura I.11 mostra um exemplo de tela da ferramenta.

Form/3 **1991 [DJA98] e [BUR98]**

É uma linguagem de programação visual que segue o paradigma utilizado nas planilhas (*spreadsheet*). Os programas são criados através da manipulação direta de células sobre a representação visual. Para cada célula define-se uma fórmula através da combinação flexível de pontos, tipos de dados e características. Possui uma coleção extensa de tipos de dados cujos atributos são definidos por fórmulas introduzidas nas células e a instanciação é representada pelo valor contido na célula. Permite a criação de novos tipos. No endereço http://www.cs.orst.edu encontram-se outras informações sobre a ferramenta.

BACCII e BACCII++ **1994 [CAL94], [CAL95] e[CAL97]**

É uma linguagem visual de programação que utiliza uma abordagem baseada em ícones, utilizada para o ensino de linguagens de programação: Pascal, C, Fortran e Basic. A versão BACCII++ é para a construção de programas em C++. As operações básicas das linguagens são representadas por ícones que são escolhidos para a construção do programa. Para cada escolha são dadas orientações de uso. Após a montagem da lógica do algoritmo, através do uso dos ícones, é gerado um arquivo fonte em uma das linguagens. Os ícones para os tipos de dados possuem representações sugestiva do que representam. No endereço http://www.csusa.net/baccii, pode-se encontrar uma versão para teste. A Figura I.12 mostra um exemplo do uso da ferramenta.

VIPERS – (Visual Programming Extensible and Reconfigurable System)
1994 [GHI98]

É um ambiente de programação visual baseado no modelo de fluxo de dado. Usa uma linguagem simples e interpretativa para definir os blocos funcionais. Estes blocos se constituem nos nós do grafo de fluxo de dados. Cada bloco corresponde a um comando (ou procedimento) que invocará a

execução de outro programa ou subprograma. Cada entrada/saída em VIPERS é caracterizado por um ícone especial indicando o tipo de dado correspondente. D-VIPERS [CAP98] é a versão para aplicações distribuídas.

Visual Craft **1997** **[HEG97]**

É um ambiente de desenvolvimento integrado que usa a programação visual e a visualização para fornecer um conjunto de ferramentas interativas para o desenvolvimento de sistemas. Composto de quatro subsistemas: analise e projeto, cenário de modelagem dos dados, programação visual e geração de código. A linguagem de programação visual baseia-se na gramática de grafos. A versão descrita gera o código em C++.

Calypso **1997** **[WOD97]**

É uma linguagem de programação visual que permite a criação de algoritmos a partir da especificação das estruturas de dados. Possui um editor de figuras para a criação externa de representações de objetos em um domínio específico. Um tradutor ligado ao editor de figuras processa as representações criadas transformando-as em representações internas. O código gerado é em Pascal.

Xing (crossing) – XML in graphics **2000** **[ERW00]**

É uma linguagem de programação visual que permite mostrar dados, consultas e programas XML de tal forma que facilite a compreensão da estrutura interna dos dados. Os elementos da linguagem XML são demonstrados em representações gráficas como caixas sobrepostas representando conjuntos de informações. Os resultados das consultas feitas sobre o conjunto de informações também são apresentados em forma gráfica e não no formato da linguagem.

8. Ferramentas que permitem a interação do estudante com o professor para correção e solução de programas

CALOS (Computer Aided Learning for Operation System) 1996 [GOL97]

É um sistema voltado ao ensino de Sistemas Operacionais. Inclui exercícios, simulações e demonstrações interativas permitindo anotações on-line. A comunicação aluno-aluno e aluno-instrutor é feita através de um *chat* e do serviço de *bulletin-board*.

ASSYST (Assessment SYSTem) 1997 [JAC97]

É uma ferramenta com uma interface gráfica para ser utilizada nos processos de avalição dos programas dos estudantes. Possui duas visões: a visão do estudante onde pode submeter seu programa para avaliação e a outra refere-se à visão do professor. Em uma janela é mostrado o código fonte do programa do estudante, os dados que ele utilizou para testar seu programa e os resultados obtidos. A ferramenta fornece ainda um relatório sobre as ocorrências do programa em relação à compilação e sobre o resultado produzido se correto ou não. Um dos principais recursos que a ferramenta possui é o de pontuar o programa do estudante. Para isso aplica métricas relativas à: correção, eficiência, estilo, complexidade e adequação do dados de teste.

ISVL (Internet Software Visualization Lab) 1997 [DOM97]

É um ambiente para visualização de software via Web. Permite a comunicação síncrona e assíncrona, não somente em ambiente de ensino à distância como também em ambientes tradicionais. A visão de um estudante pode ser recuperada, por exemplo, por todos os demais estudantes da classe. Cada um deles pode registrar, num determinado exemplo apresentado à eles, seus comentários e suas sugestões. Essas interferências no problema são carregadas e redistribuídas à todos. As animações são produzidas através da inclusão de “eventos de interesse” dentro do algoritmo, como em BALSA [BRO84].

PILOT (Platform-Independent Learning Online Tool) 2000 [BRI00]

É um sistema interativo para a web, utilizado para testar conceitos de ciência da computação através de recursos visuais. Os problemas implementados na ferramenta estão relacionados com árvores e grafos. Escolhido um problema o estudante interage com ele propondo uma solução. Essa solução pode ser submetida para avaliação. Da avaliação é fornecida uma pontuação e a solução correta para o problema. Usa arquitetura cliente-servidor e as demonstrações das soluções são efetuadas por *applets*.

9. Outros ambientes com Interfaces Gráficas para: especificação de programas, sistema para programação genética, processamento de linguagem natural, etc.

HandMove 1995 [VOD95] e [VOD97]

É um sistema que cria elementos animados independentes para serem integrados num sistema construtor de interfaces. As animações construídas podem ser representações espontâneas interativas ou aplicações controladas. HandMove organiza a animação em cenas onde são montados os objetos gráficos. O modelo baseia-se em animação orientada à caminhos, dirigido por sinais de tempo, valores de aplicação e entradas efetuadas pelo usuário. HandMove possui uma linguagem de programação visual que cria a semântica da interface criada.

PROGRES (PROgramming with Graph Rewriting System) 1995 [SCH95]

É uma linguagem visual para especificação de programa. Com ela o usuário constrói a especificação de seu problema, usando objetos gráfico cuja representação combina diagramas de classes da orientação à objetos com regras de grafos, estabelecendo relações entre entidades. A validação da especificação é garantida através de um editor dirigido à sintaxe, um parser e recurso para verificação de pré e pós condições. A proposta futura é que a representação gráfica utilize a notação de diagrama de classes da UML

(*Unified Modelling Language*) [MÜN98]. Para especificar as ações das regras utiliza uma linguagem textual simples que é parte do sistema. No endereço http://www-i3.informatik.rwth-aachen.de/research/projects/progres pode-se obter outras informações e uma cópia de PROGRES. A Figura I.13 apresenta o exemplo de uma tela com uma aplicação e PROGRES.

VZ (Visual Z) 1997 [CHA97]

É um ambiente de programação visual para construção de especificações em Z. A interface é gráfica e possui ícones e botões para a escolha dos elementos para a composição da especificação. É possível fazer a validação da especificação e produzir uma versão textual da especificação.

EqEditor (Equation Editor) 1997 [JUM97]

É uma ferramenta que permite ao usuário editar expressões matemáticas na *web*. É indicada para ser utilizada como interface de entrada para algum sistema que envolva álgebra. Trata-se de uma *applet* escrita em Java.

PUIST (Petri net based graphical User Interface Specification Tool) 1997 [LI 97]

É uma linguagem visual para a construção de protótipos e especificação de interfaces gráficas. Utiliza notação de redes de Petri, onde os nós, por exemplo, representam objetos da GUI, tais como janelas ou itens do menu. Possui uma interface gráfica onde é possível “desenhar ” os componentes da interface que o usuário deseja criar;

Amulet (Automatic Manufacture of Usable and Learnable Editors and Toolkits) 1997 [MYE97]

É um ambiente visual de desenvolvimento de interface gráfica para C++. Propõe ajudar o usuário no projeto, na criação do protótipo, na implementação e na avaliação de interface do usuário. Amulet possui uma interface gráfica que permite a criação de interfaces de forma interativa e também a manipulação direta dos elementos criados. Cada elemento é

representado como um objeto. É portável para todo tipo de plataforma: Unix, Microsoft Windows 95 e NT e Macintosh.

SAM (Solid Agents in Motion) 1998 [GEI98]

É uma linguagem de programação visual para a especificação e animação (em 3D) de sistemas paralelos. Agentes e mensagens possuem uma representação abstrata e uma concreta em 3D. A especificação de mensagens, agentes, regras e ações são dadas como texto e podem ser acessadas através de formulários que aparecem quando recebem um duplo clique sobre o objeto em 3D.

VDBMS (Visual Data Base Management System) 1998 [SEB98]

É um ambiente visual para desenvolvimento de sistemas de base de dados. Usa recursos visuais para dar suporte à todas as fases do ciclo de vida de um sistema de base de dados. No ambiente os dados, regras de integridade, recuperações de dados e seus resultados, são processados como sentenças definidas e representadas por ícones.

JPT (Java Power Tools) 1999 [RAS01]

É uma ferramenta visual para auxiliar no projeto de interfaces gráficas voltadas ao usuário. JPT baseia-se no paradigma do modelo controlador de visões. Neste paradigma o modelo interno de dados é separado das visões da interface que mostram ou demonstram os dados. Suporta três níveis do projeto de interface de usuário: o modelo de console tradicional de entrada e saída; caixas de diálogos simples para entrada de tipos básicos e interfaces gráficas completas. Utiliza a linguagem Java para a construção dos projetos. JPT foi empregada basicamente no ensino do paradigma de orientação à objetos pois declara que a ferramenta consegue demonstrar a potência da programação orienta à objetos ao mesmo tempo em que simplifica a criação das interfaces gráficas. No endereço www.ccs.neu.edu/teaching/EdGroup, pode-se encontrar exemplos e outros elementos da ferramenta.

GAPS **2000 [KRA00]**

É um sistema para programação genético que implementa o protótipo de um algoritmo geral para programação genética. Inclui uma linguagem extensível para as necessidades da programação genética. É um sistema completo que permite que a tarefa de programação genética seja concluída sem requerer outras ferramentas, tais como um compilador. Possui uma interface visual pois foi desenvolvido num ambiente de janelas em C++.

CAILS (Computer Assisted Iconic Language System) **2000 [CHA00]**

É uma linguagem visual para processamento de linguagem natural. Utiliza um método baseado em ícones, onde os ícones são usados para representar unidades léxicas. Foi projetado para a comunicação humano – humano. Através da combinação de símbolos elementares (gramática) com figuras (vocabulário), produz ícones significativos para atuar na comunicação (a unidade de expressão). Os ícones são selecionado a partir do conjunto de vocábulos representados por ícones e então são associados à uma função gramatical. Os vocábulos escolhidos referem-se às 850 palavras em inglês, sugeridas por Ogden, como sendo o conjunto necessário para a comunicação básica.

DIAPLAN **2000 [HOF00]**

É uma linguagem visual baseada em regras e no modelo computacional de transformações de grafos. Consiste de uma linguagem que oferece recursos para edição e estruturação do grafo. Pode gerar ambientes visuais sem restrições sobre a representação visual e possui recursos para a programação orientada à objetos desde que os grafos sejam estruturados hierarquicamente. Para isso ele oferece recursos para abstração, controle e encapsulamento. Ele é formado por um editor visual de programas, um compilador e um interpretador.

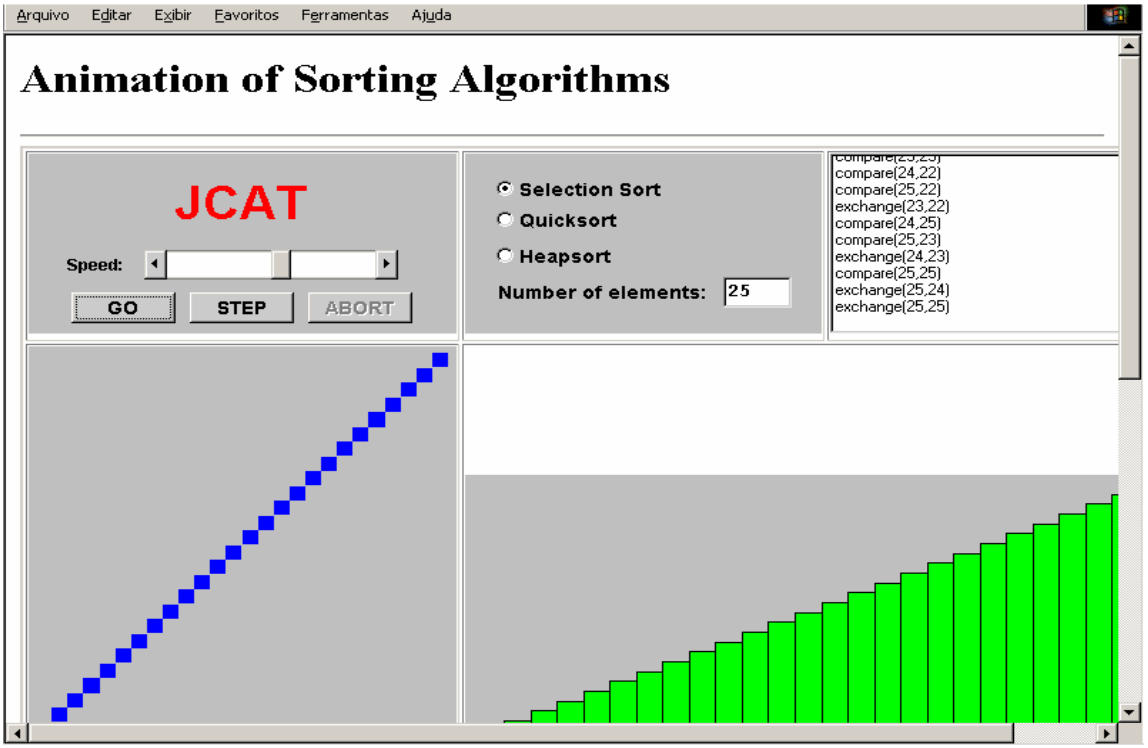


Figura I.1 Exemplo de Tela do JCAT

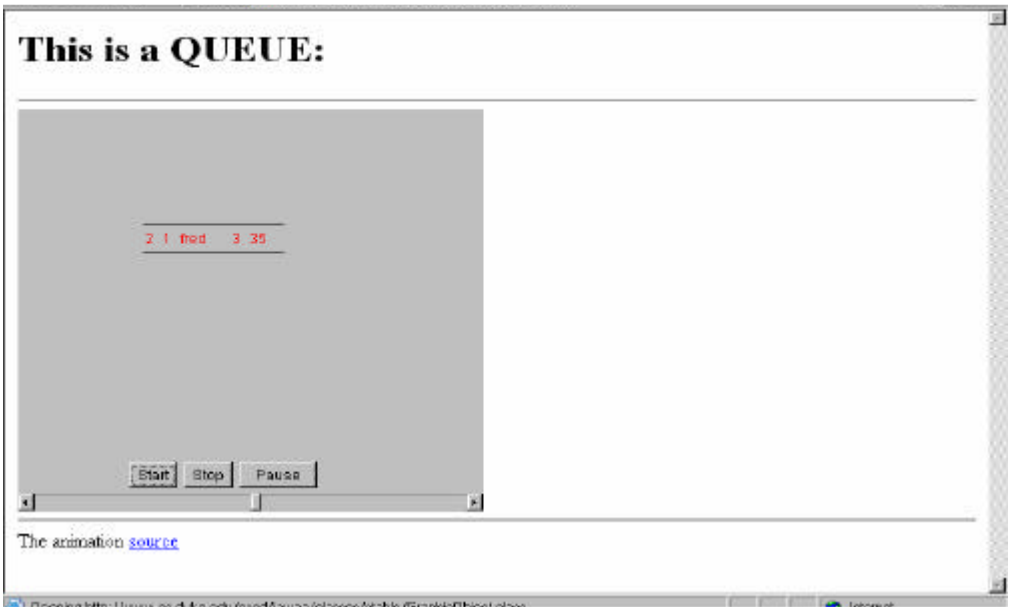


Figura I.2 – Tela da Ferramenta JAWAA

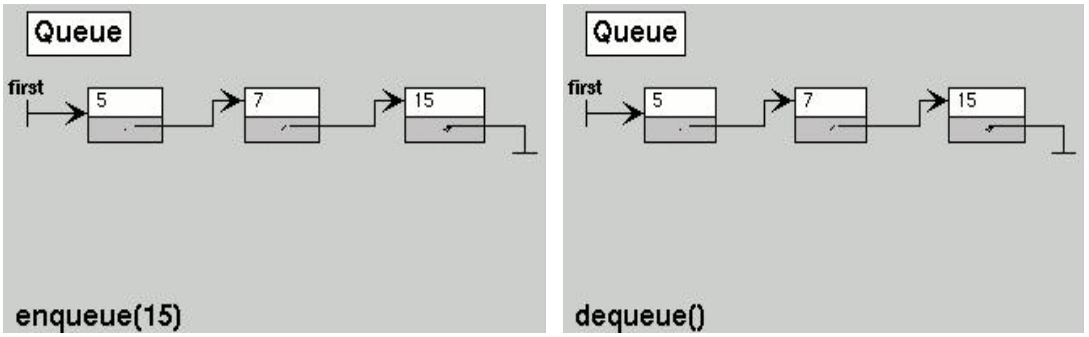


Figura I.3 – Exemplo do uso de AnimalScript

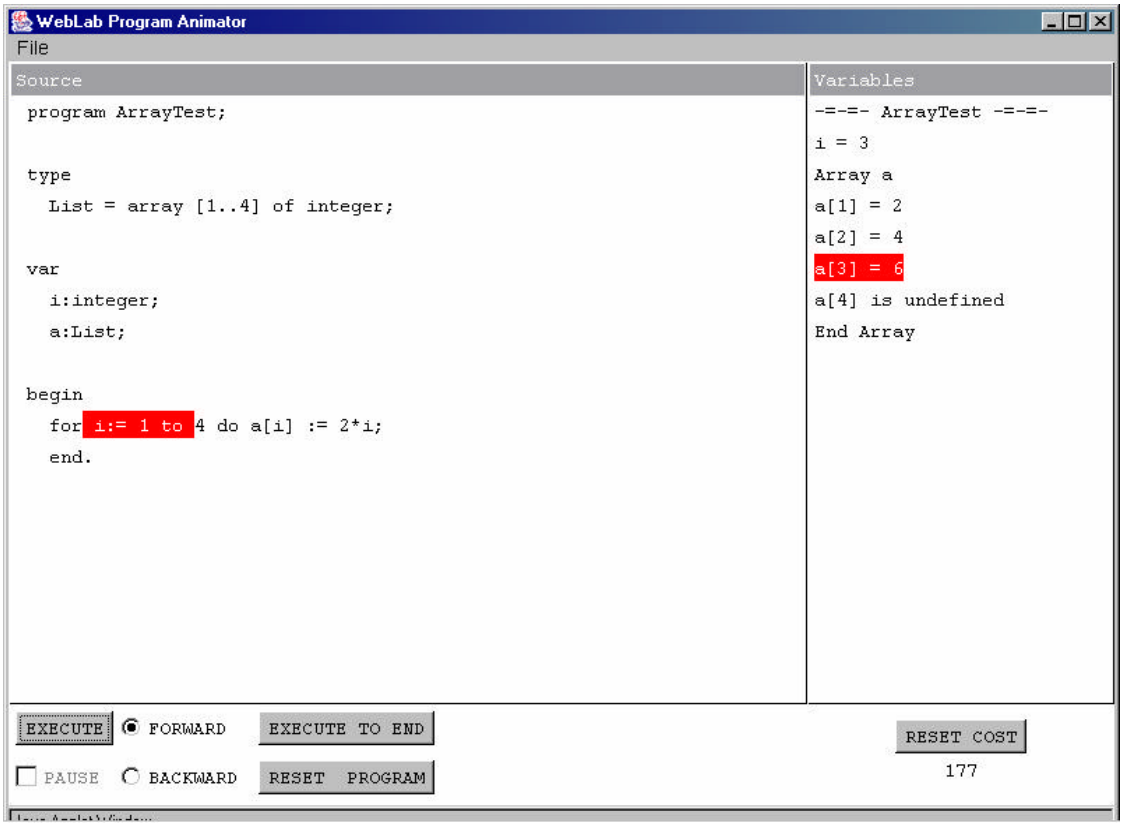


Figura I.4 – Animação construída em Dynalab sobre um vetor

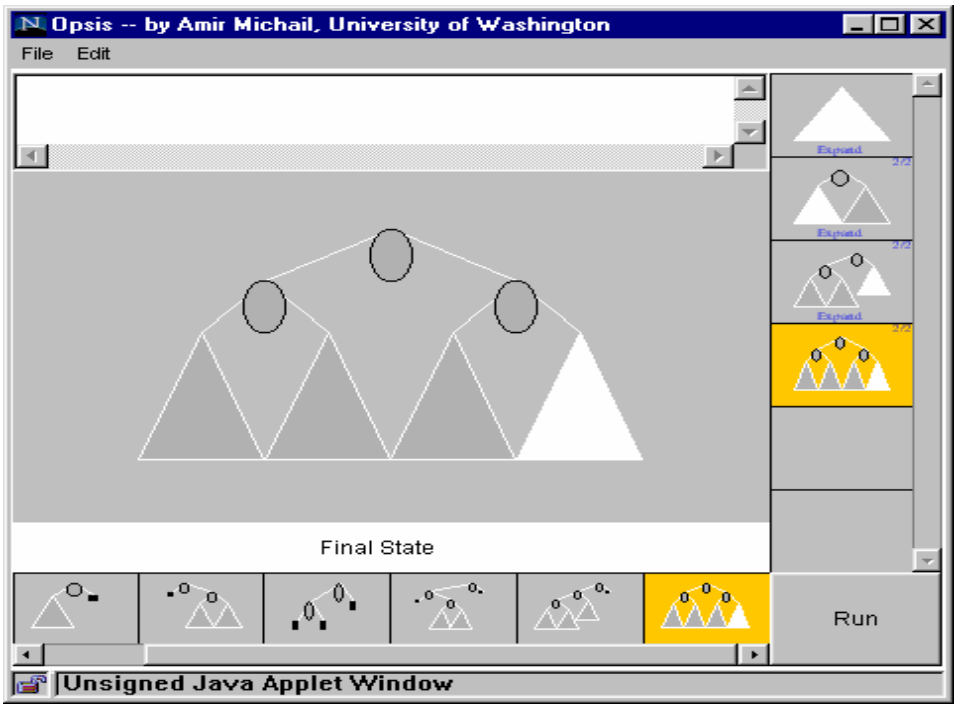


Figura I.5 – Exemplo de Opsis

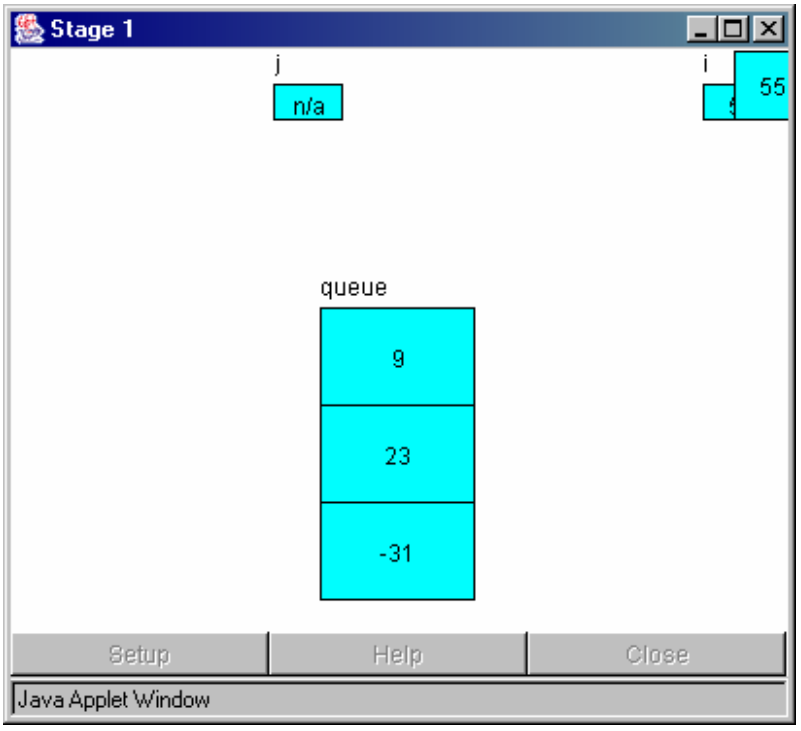


Figura I.6 – Exemplo de tela de Jeliot

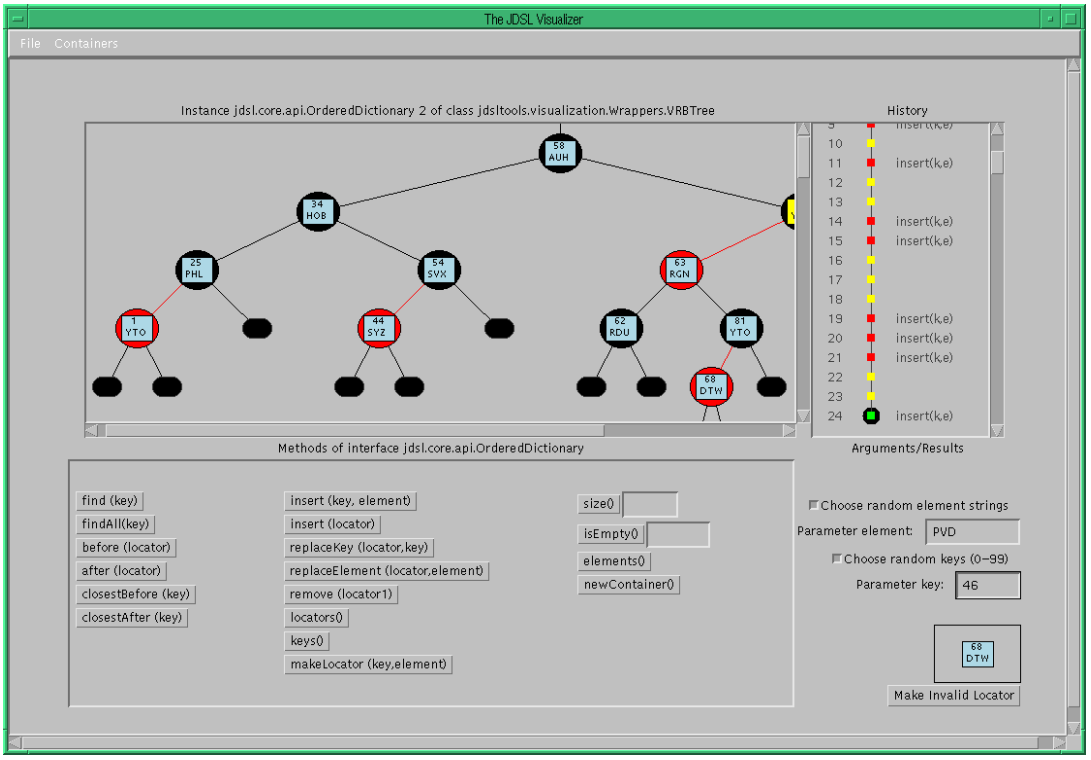


Figura I.7 – Tela da JDSL

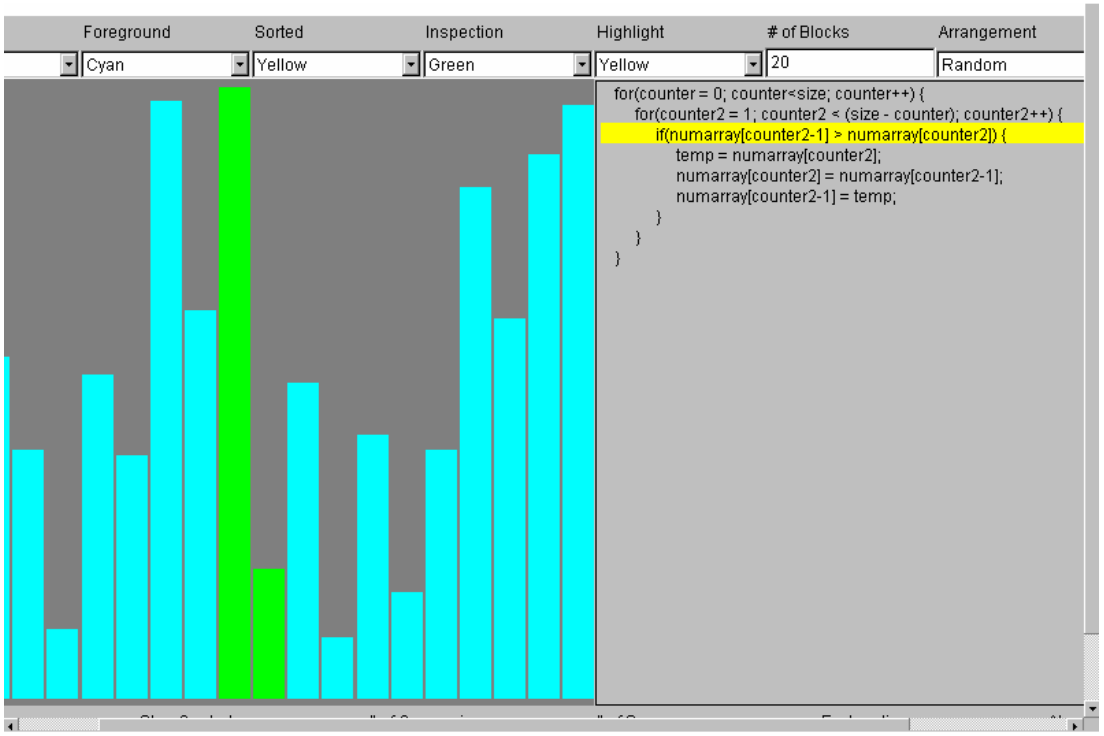


Figura I.8 – Tela da ferramenta Animator

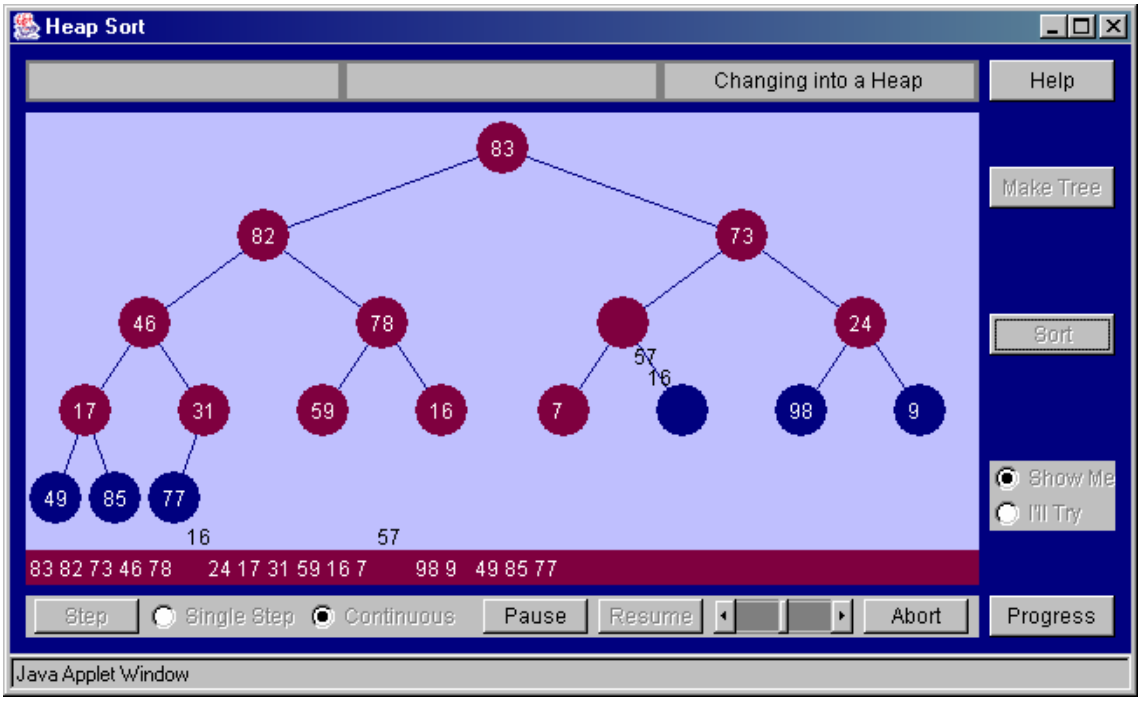


Figura I.9 – Exemplo de IDSV

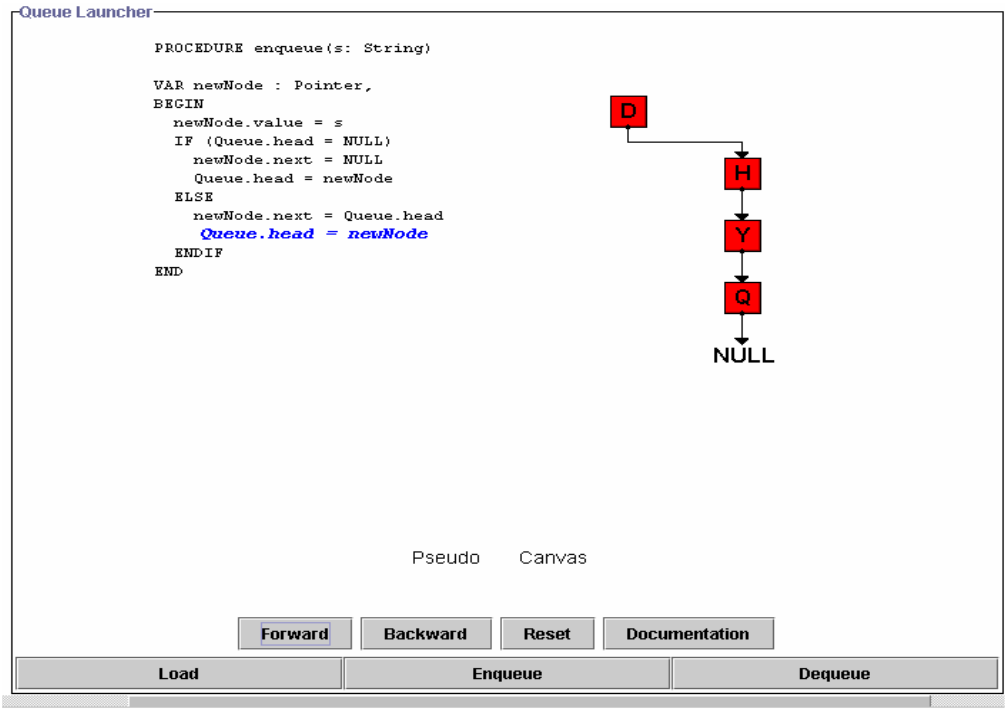


Figura I.10 – Exemplo de Algorithma98

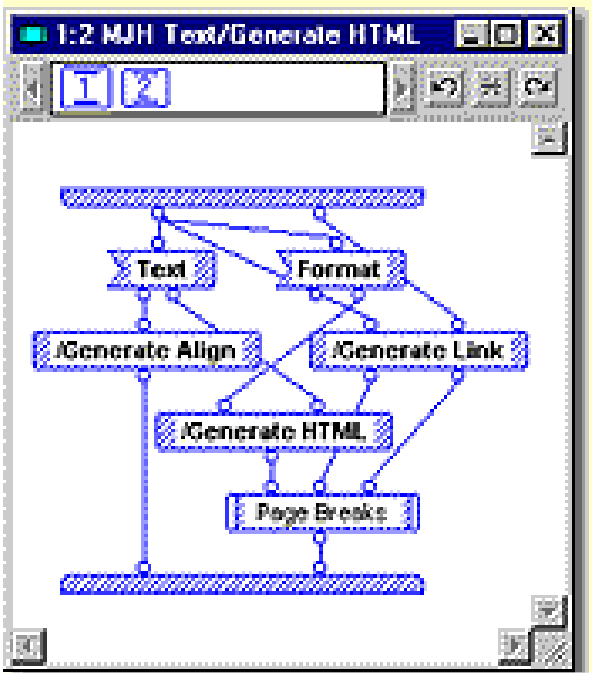


Figura I.11 – Exemplo de uso do Prograph

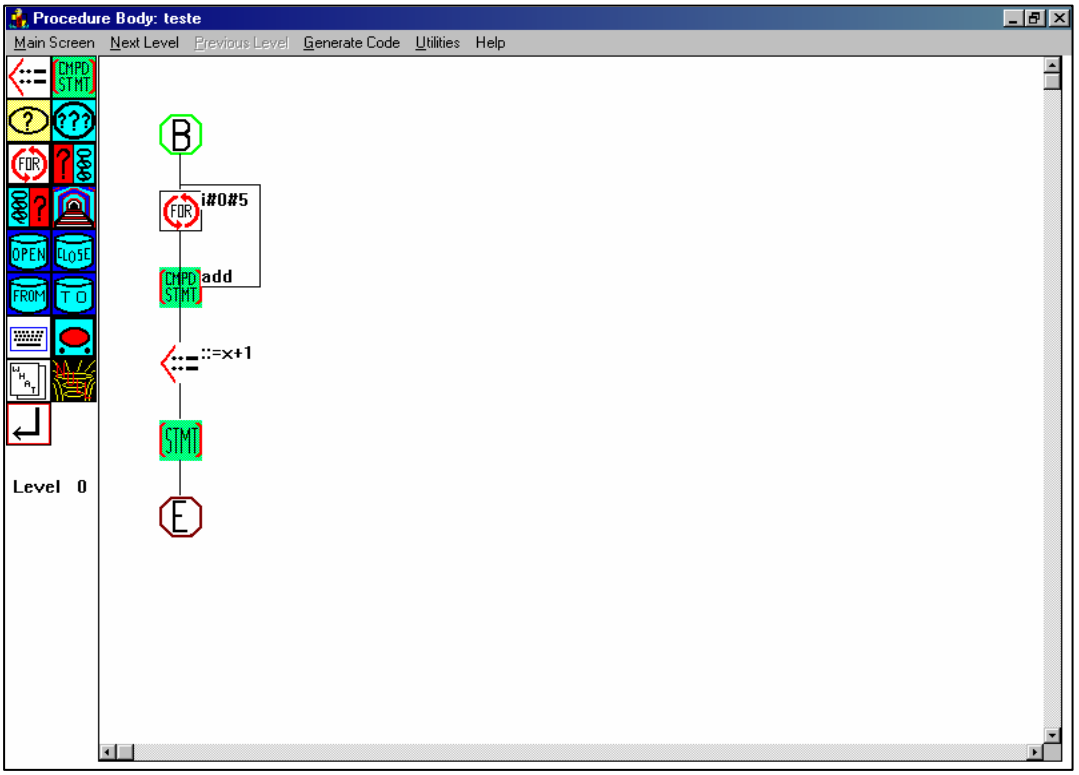


Figura I.12 – Exemplo de uma tela de BACCII++

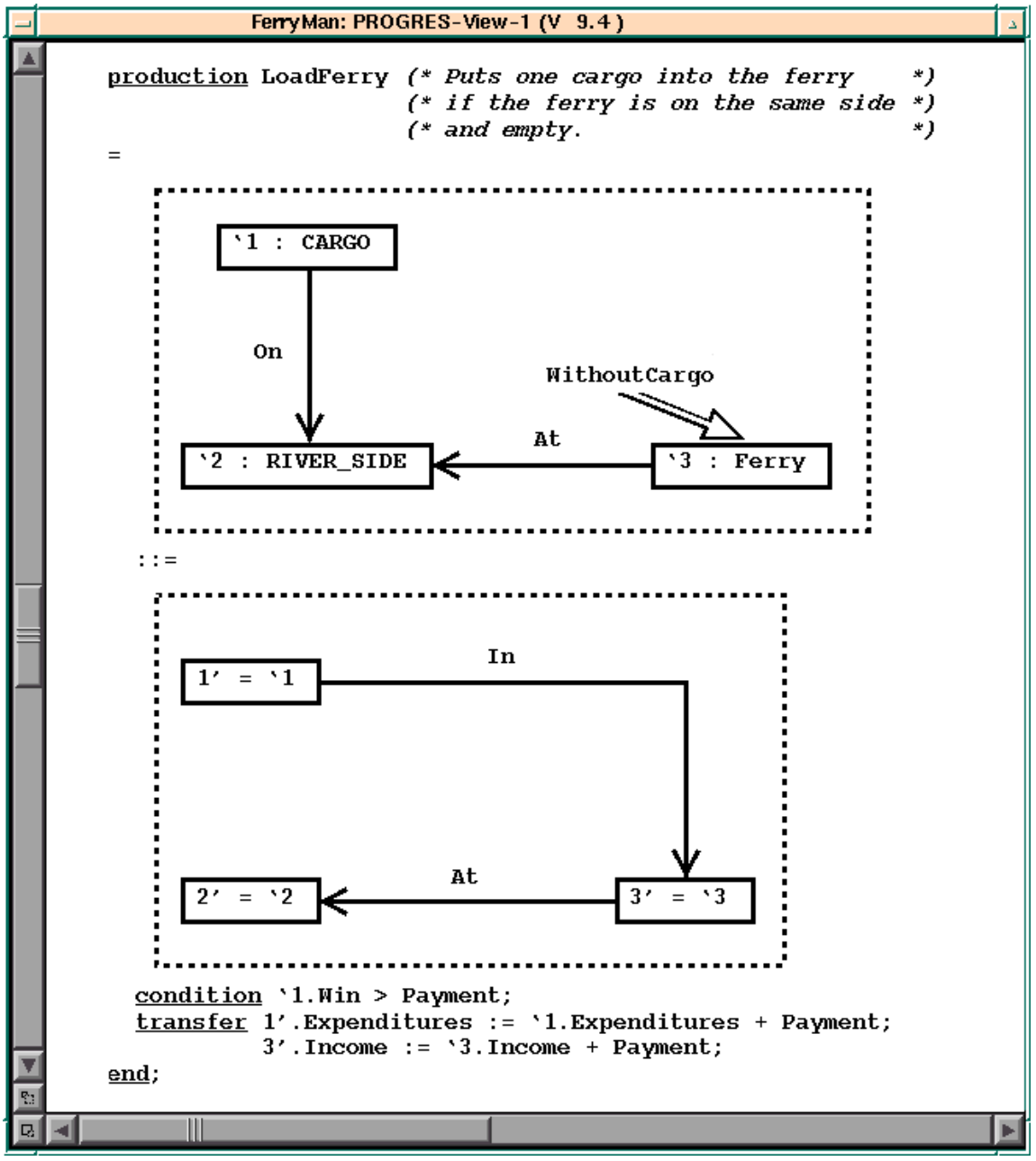


Figura I.13 – Exemplo de Tela de Progress

ANEXO II

Introdução

A partir do dígrafo de chamadas, apresentado na Figura 6.3, foram gerados os caminhos sem repetições, de comprimento menor do que 7. Para isso levou-se em consideração os conceitos de teoria de grafos [BER75].

Definição II.2: um grafo direcionado, dígrafo ou grafo orientado é o par ordenado $D = \langle A, R \rangle$, onde A é um conjunto não vazio de nós e R é uma relação em A, isto é, R é um conjunto de pares ordenados, que são chamados de arcos.

Os botões que pertencem à interface da ADTTool compõem o conjunto A e a relação lógica de chamadas que produzem um resultado significativo representam a relação R. Considerando-se a seqüência numérica para cada um dos botões como descrito no capítulo 6, obtém-se:

$A = \{ 1, 2, 3, 4, 5, 6, 7 \}$

$R = \{ \langle 1,2 \rangle, \langle 1,7 \rangle, \langle 2,2 \rangle, \langle 2,3 \rangle, \langle 2,4 \rangle, \langle 2,7 \rangle, \langle 3,2 \rangle, \langle 3,5 \rangle, \langle 3,7 \rangle, \langle 4,5 \rangle, \langle 4,6 \rangle, \langle 4,7 \rangle, \langle 5,2 \rangle, \langle 5,3 \rangle, \langle 5,4 \rangle, \langle 5,5 \rangle, \langle 5,7 \rangle, \langle 6,2 \rangle, \langle 6,3 \rangle, \langle 6,4 \rangle, \langle 6,5 \rangle, \langle 6,7 \rangle \}$

Definição II.2: Seja $D = \langle A, R \rangle$ um dígrafo, com $A = \{ a_1, a_2, a_3, \ldots, a_n \}$. A Matriz de Adjacência X de D é definida por:

$x_{ij} = 1$ se $\langle a_i, a_j \rangle \in R$,

$x_{ij} = 0$ se $\langle a_i, a_j \rangle \notin R$

A matriz de adjacências X do dígrafo que representa os botões da interface é dada por:

$$X = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Definição II.3: Seja D = <A,R> um dígrafo, com $A = \{ a_1, a_2, a_3, \ldots, a_n \}$. A matriz de caminhos P de D é definida por:

$p_{ij} = 1$ se $\exists$ um caminho não nulo de a_i até a_j;

$p_{ij} = 0$ se $\nexists$ um caminho não nulo de a_i até a_j;

Teorema II.1: Seja X a matriz de adjacência do dígrafo D, e seja $Y = X^h$. Então y_{ij} representa o número total de caminhos distintos em D que tem comprimento h.

Corolário II.1(do teorema II.1): $P = X \vee X^2 \vee X^3 \vee \ldots \vee X^n$, onde

$X^k = X^{k-1} \wedge X$ e, $X^{k-1} \wedge X$ representa o produto lógico da matriz X^{k-1} pela matriz X.

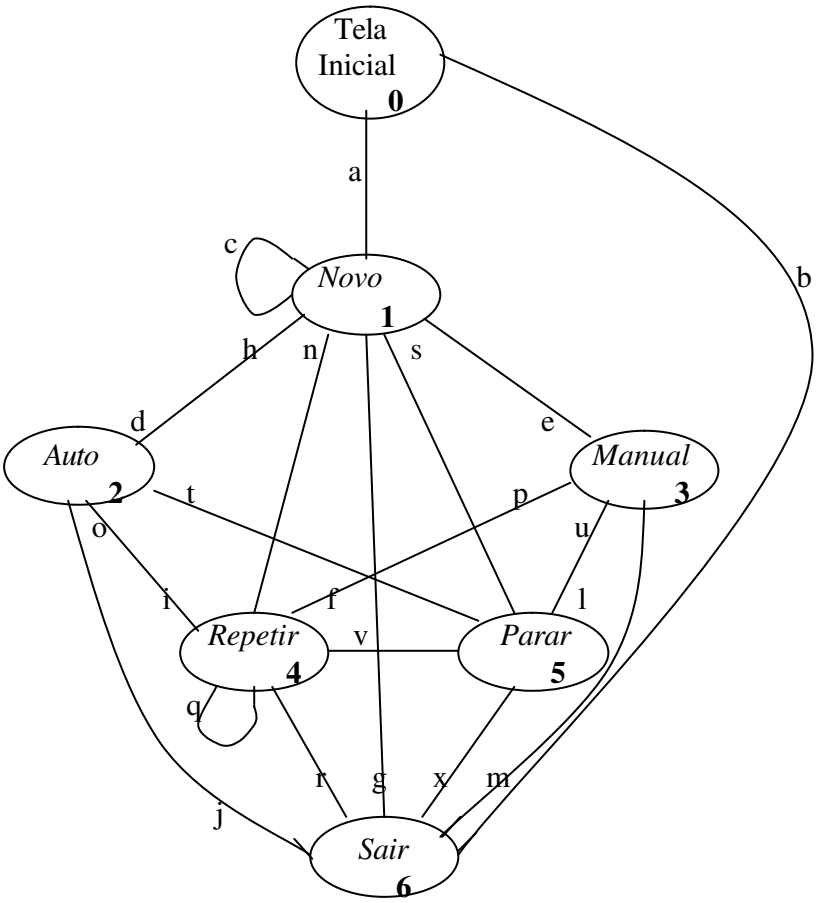
Assim sendo, aplicando-se o produto lógico na matriz de adjacência que representa a interface tem-se, para k = 2:

$$X^2 = X \wedge X = \begin{pmatrix} 0 & 1 & 1 & 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

A matriz resultante traz os caminhos existentes de comprimento 2, com ciclos ou não. Por exemplo, como o elemento da matriz na linha 1 coluna 3 é igual a 1, significa que existe pelo menos um caminho de comprimento 2 do nó 1 até o nó 3.

Entretanto não está visível na matriz quais são estes caminhos e se existe mais de um caminho. Isto é, não está claro quais os arcos que participam de cada caminho. Assim sendo, ao atribuir rótulos aos arcos existentes no dígrafo o resultado do produto lógico da matriz trará em cada posição onde existe um caminho de um certo comprimento h, os arcos que participam daquele caminho.

O dígrafo que representa a interface com rótulos fica:



A matriz de adjacência correspondente ao dígrafo da interface com os rótulos nos arcos é dada por:

$$X = \begin{pmatrix} 0 & a & 0 & 0 & 0 & 0 & b \\ 0 & c & d & e & 0 & 0 & g \\ 0 & h & 0 & 0 & i & 0 & j \\ 0 & 0 & 0 & 0 & f & l & m \\ 0 & n & o & p & q & 0 & r \\ 0 & s & t & u & v & 0 & x \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Ao efetuar algumas vezes o produto lógico sobre esta matriz tem-se: $X^2 = X \wedge X$

$$X^2 = \begin{pmatrix} 0 & ac & ad & ae & 0 & 0 & ag \\ 0 & cc\ ,\ dh & cd & ce & di\ ,\ ef & el & cg,\ dj,\ em \\ 0 & hc,\ in & hd,\ io & he,\ ip & iq & 0 & hg,\ ir \\ 0 & fn,\ ls & fo,\ lt & fp,\ lu & fq,\ lv & 0 & fr,\ lx \\ 0 & nc,oh,qn & nd,qo & ne,qp & oi,pf,qq & pl & ng,oj,pm,qr \\ 0 & sc,th,vn & sd,vo & se,vp & ti,uf,vq & ul & sg,tj,um,vr \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

$X^3 = X^2 \wedge X$, matriz contendo os caminhos de comprimento 3 é dada por

$$X^3 = \begin{pmatrix}
0 & \text{acc,adh} & \text{acd} & \text{ace} & \text{adi,aef} & \text{ael} & \text{acg,adj,aem} \\
0 & \text{ccc,dhc,cdh,din,efn,els} & \text{ccd,dhd,dio,efo,elt} & \text{cce,dhe,dip,efp,elu} & \text{cdi,cef,diq,efq,elv} & \text{cel} & \text{ccg,dhg,cdj,cem,dir,efr,elx} \\
0 & \text{hcc,inc,hdh,ioh,iqn} & \text{hcd,ind,iqo} & \text{hce,ine,iqp} & \text{hdi.ioi,hef,ipf,iqq} & \text{hel,ipl} & \text{hcg,ing,hdj,ioj,hem,ipm,iqr} \\
0 & \text{fnc,lsc,foh,lth,fqn,lvn} & \text{fnd,lsd,fqo,lvo} & \text{fne,lse,fqp,lvp} & \text{foi,lti,fpf,luf,fqq,lvq} & \text{fpl,lul} & \text{fng,lsg,foj,ltj,fpm,lum,fqr,lvr} \\
0 & \text{ncc,ohc,qnc,ndh,qoh,oin,pfn,qqn,pls} & \text{ncd,ohd,qnd,oio,pfo,qqo,plt} & \text{nce,ohe,qne,oip,pfp,qqp,plu} & \text{ndi,qoi,nef,qpf,oiq,pfq,qqq,plv} & \text{nel,qpl,} & \text{ncg,ohg,qng,ndj,qojnem,qpm,oir,pfr,qqr,plx} \\
0 & \text{scc,thc,vnc,sdh,voh,tin,ufn,vqn,uls} & \text{scd,thd,vnd,tio,ufo,vqo,ult} & \text{sce,the,vne,tip,ufp,vqp,ulu} & \text{sdi,voi,sef,vpf,tiq,ufq,vqq,ulv} & \text{sel,vpl} & \text{scg,thg,vng,sdj,voj,sevpm,tir,ufr,vqr,ulx} \\
0 & 0 & 0 & 0 & 0 & 0 & 0
\end{pmatrix}$$

Observando-se a matriz X^3, o conteúdo da posição relativa à linha 2 e coluna 7 é dado por: *ccg, dhg, cdj, cem, dir, efr, elx*. Significa que existem 7 caminhos de comprimento 3, saindo do nó 2 e chegando no nó 7. As letras representam os arcos dos caminhos.

Com base neste conceito foi criado um programa para determinar todos os caminhos de comprimento *h*, com $2 \leq h \leq n$, onde n é o número de nós do dígrafo. Caminhos com comprimento maior do que o número de nós resultarão na repetição de nós. De todos os caminhos encontrados, o programa seleciona apenas aqueles que não possuem ciclos, isto é, sem repetições de nós. A listagem do programa consta do relatório técnico [ENG02].

A seguir é apresentada a relação completa, gerada pelo programa, de todos os caminhos sem repetições que existem no dígrafo que representa a interface. Estes caminhos representam as possibilidades de escolha dos botões da interface e que resultarão numa ação válida.

Caminhos sem Repetições

```
------------------------------------------------

Tela Inicial --> Novo
Tela Inicial --> Novo --> Auto
Tela Inicial --> Novo --> Manual  --> Repetir --> Auto
Tela Inicial --> Novo --> Manual  --> Parar   --> Auto
Tela Inicial --> Novo --> Manual  --> Parar   --> Repetir --> Auto
Tela Inicial --> Novo --> Manual
Tela Inicial --> Novo --> Auto    --> Repetir --> Manual
Tela Inicial --> Novo --> Auto    --> Repetir
Tela Inicial --> Novo --> Manual  --> Repetir
Tela Inicial --> Novo --> Manual  --> Parar   --> Repetir
Tela Inicial --> Novo --> Manual  --> Parar   --> Auto    --> Repetir
Tela Inicial --> Novo --> Manual  --> Parar
Tela Inicial --> Novo --> Auto    --> Repetir --> Manual  --> Parar
Tela Inicial --> Sair
Tela Inicial --> Novo --> Sair
Tela Inicial --> Novo --> Auto    --> Sair
Tela Inicial --> Novo --> Manual  --> Sair
Tela Inicial --> Novo --> Auto    --> Repetir --> Sair
Tela Inicial --> Novo --> Manual  --> Repetir --> Sair
Tela Inicial --> Novo --> Manual  --> Parar   --> Sair
Tela Inicial --> Novo --> Manual  --> Repetir --> Auto    --> Sair
Tela Inicial --> Novo --> Manual  --> Parar   --> Auto    --> Sair
Tela Inicial --> Novo --> Auto    --> Repetir --> Manual  --> Sair
Tela Inicial --> Novo --> Manual  --> Parar   --> Repetir --> Sair
Tela Inicial --> Novo --> Manual  --> Parar   --> Repetir --> Auto  -->
Sair
```

```
Tela Inicial --> Novo --> Manual  --> Parar   --> Auto  --> Repetir -->
Sair
Tela Inicial --> Novo --> Auto    --> Repetir --> Manual --> Parar  -->
Sair

Novo --> Auto
Novo --> Manual  --> Repetir --> Auto
Novo --> Manual  --> Parar   --> Auto
Novo --> Manual  --> Parar   --> Repetir --> Auto
Novo --> Manual
Novo --> Auto    --> Repetir --> Manual
Novo --> Auto    --> Repetir
Novo --> Manual  --> Repetir
Novo --> Manual  --> Parar   --> Repetir
Novo --> Manual  --> Parar   --> Auto    --> Repetir
Novo --> Manual  --> Parar
Novo --> Auto    --> Repetir --> Manual  --> Parar
Novo --> Sair
Novo --> Auto    --> Sair
Novo --> Manual  --> Sair
Novo --> Auto    --> Repetir --> Sair
Novo --> Manual  --> Repetir --> Sair
Novo --> Manual  --> Parar   --> Sair
Novo --> Manual  --> Repetir --> Auto    --> Sair
Novo --> Manual  --> Parar   --> Auto    --> Sair
Novo --> Auto    --> Repetir --> Manual  --> Sair
Novo --> Manual  --> Parar   --> Repetir --> Sair
Novo --> Manual  --> Parar   --> Repetir --> Auto    --> Sair
Novo --> Manual  --> Parar   --> Auto    --> Repetir --> Sair
Novo --> Auto    --> Repetir --> Manual  --> Parar   --> Sair

Auto --> Novo
Auto --> Repetir --> Novo
Auto --> Repetir --> Manual  --> Parar   --> Novo
Auto --> Novo    --> Manual
Auto --> Repetir --> Manual
Auto --> Repetir --> Novo    --> Manual
Auto --> Repetir
Auto --> Novo    --> Manual  --> Repetir
Auto --> Novo    --> Manual  --> Parar   --> Repetir
Auto --> Novo    --> Manual  --> Parar
Auto --> Repetir --> Manual  --> Parar
Auto --> Repetir --> Novo    --> Manual  --> Parar
Auto --> Sair
Auto --> Novo    --> Sair
Auto --> Repetir --> Sair
Auto --> Repetir --> Novo    --> Sair
Auto --> Novo    --> Manual  --> Sair
Auto --> Repetir --> Manual  --> Sair
Auto --> Repetir --> Novo    --> Manual  --> Sair
Auto --> Novo    --> Manual  --> Repetir --> Sair
Auto --> Novo    --> Manual  --> Parar   --> Sair
Auto --> Repetir --> Manual  --> Parar   --> Sair
Auto --> Repetir --> Manual  --> Parar   --> Novo    --> Sair
Auto --> Novo    --> Manual  --> Parar   --> Repetir --> Sair
Auto --> Repetir --> Novo    --> Manual  --> Parar   --> Sair
```

```
Manual --> Repetir --> Novo
Manual --> Parar   --> Novo
Manual --> Repetir --> Auto    --> Novo
Manual --> Parar   --> Auto    --> Novo
Manual --> Parar   --> Repetir --> Novo
Manual --> Parar   --> Repetir --> Auto    --> Novo
Manual --> Parar   --> Auto    --> Repetir --> Novo
Manual --> Repetir --> Auto
Manual --> Parar   --> Auto
Manual --> Repetir --> Novo    --> Auto
Manual --> Parar   --> Novo    --> Auto
Manual --> Parar   --> Repetir --> Auto
Manual --> Parar   --> Repetir --> Novo    --> Auto
Manual --> Repetir
Manual --> Parar   --> Repetir
Manual --> Parar   --> Auto    --> Repetir
Manual --> Parar   --> Novo    --> Auto    --> Repetir
Manual --> Parar
Manual --> Sair
Manual --> Repetir --> Sair
Manual --> Parar   --> Sair
Manual --> Repetir --> Novo    --> Sair
Manual --> Parar   --> Novo    --> Sair
Manual --> Repetir --> Auto    --> Sair
Manual --> Parar   --> Auto    --> Sair
Manual --> Parar   --> Repetir --> Sair
Manual --> Repetir --> Auto    --> Novo    --> Sair
Manual --> Parar   --> Auto    --> Novo    --> Sair
Manual --> Parar   --> Repetir --> Novo    --> Sair
Manual --> Repetir --> Novo    --> Auto    --> Sair
Manual --> Parar   --> Novo    --> Auto    --> Sair
Manual --> Parar   --> Repetir --> Auto    --> Sair
Manual --> Parar   --> Auto    --> Repetir --> Sair
Manual --> Parar   --> Repetir --> Auto    --> Novo    --> Sair
Manual --> Parar   --> Auto    --> Repetir --> Novo    --> Sair
Manual --> Parar   --> Repetir --> Novo    --> Auto    --> Sair
Manual --> Parar   --> Novo    --> Auto    --> Repetir --> Sair

Repetir --> Novo
Repetir --> Auto    --> Novo
Repetir --> Manual  --> Parar   --> Novo
Repetir --> Manual  --> Parar   --> Auto    --> Novo
Repetir --> Auto
Repetir --> Novo    --> Auto
Repetir --> Manual  --> Parar   --> Auto
Repetir --> Manual  --> Parar   --> Novo    --> Auto
Repetir --> Novo    --> Manual  --> Parar   --> Auto
Repetir --> Manual
Repetir --> Novo    --> Manual
Repetir --> Auto    --> Novo    --> Manual
Repetir --> Manual  --> Parar
Repetir --> Novo    --> Manual  --> Parar
Repetir --> Auto    --> Novo    --> Manual  --> Parar
Repetir --> Sair
Repetir --> Novo    --> Sair
Repetir --> Auto    --> Sair
Repetir --> Manual  --> Sair
```

```
Repetir --> Auto    --> Novo    --> Sair
Repetir --> Novo    --> Auto    --> Sair
Repetir --> Novo    --> Manual  --> Sair
Repetir --> Manual  --> Parar   --> Sair
Repetir --> Manual  --> Parar   --> Novo    --> Sair
Repetir --> Manual  --> Parar   --> Auto    --> Sair
Repetir --> Auto    --> Novo    --> Manual  --> Sair
Repetir --> Novo    --> Manual  --> Parar   --> Sair
Repetir --> Manual  --> Parar   --> Auto    --> Novo    --> Sair
Repetir --> Manual  --> Parar   --> Novo    --> Auto    --> Sair
Repetir --> Novo    --> Manual  --> Parar   --> Auto    --> Sair
Repetir --> Auto    --> Novo    --> Manual  --> Parar   --> Sair

Parar --> Novo
Parar --> Auto    --> Novo
Parar --> Repetir --> Novo
Parar --> Repetir --> Auto    --> Novo
Parar --> Auto    --> Repetir --> Novo
Parar --> Manual  --> Repetir --> Novo
Parar --> Manual  --> Repetir --> Auto    --> Novo
Parar --> Auto
Parar --> Novo    --> Auto
Parar --> Repetir --> Auto
Parar --> Repetir --> Novo    --> Auto
Parar --> Manual  --> Repetir --> Auto
Parar --> Manual  --> Repetir --> Novo    --> Auto
Parar --> Novo    --> Manual  --> Repetir --> Auto
Parar --> Manual
Parar --> Novo    --> Manual
Parar --> Repetir --> Manual
Parar --> Auto    --> Novo    --> Manual
Parar --> Repetir --> Novo    --> Manual
Parar --> Auto    --> Repetir --> Manual
Parar --> Repetir --> Auto    --> Novo    --> Manual
Parar --> Auto    --> Repetir --> Novo    --> Manual
Parar --> Novo    --> Auto    --> Repetir --> Manual
Parar --> Repetir
Parar --> Auto    --> Repetir
Parar --> Manual  --> Repetir
Parar --> Novo    --> Auto    --> Repetir
Parar --> Novo    --> Manual  --> Repetir
Parar --> Auto    --> Novo    --> Manual  --> Repetir
Parar --> Sair
Parar --> Novo    --> Sair
Parar --> Auto    --> Sair
Parar --> Manual  --> Sair
Parar --> Repetir --> Sair
Parar --> Auto    --> Novo    --> Sair
Parar --> Repetir --> Novo    --> Sair
Parar --> Novo    --> Auto    --> Sair
Parar --> Repetir --> Auto    --> Sair
Parar --> Novo    --> Manual  --> Sair
Parar --> Repetir --> Manual  --> Sair
Parar --> Auto    --> Repetir --> Sair
Parar --> Manual  --> Repetir --> Sair
Parar --> Repetir --> Auto    --> Novo    --> Sair
Parar --> Auto    --> Repetir --> Novo    --> Sair
```

```
Parar --> Manual  --> Repetir --> Novo    --> Sair
Parar --> Repetir --> Novo    --> Auto    --> Sair
Parar --> Manual  --> Repetir --> Auto    --> Sair
Parar --> Auto    --> Novo    --> Manual  --> Sair
Parar --> Repetir --> Novo    --> Manual  --> Sair
Parar --> Auto    --> Repetir --> Manual  --> Sair
Parar --> Novo    --> Auto    --> Repetir --> Sair
Parar --> Novo    --> Manual  --> Repetir --> Sair
Parar --> Manual  --> Repetir --> Auto    --> Novo    --> Sair
Parar --> Manual  --> Repetir --> Novo    --> Auto    --> Sair
Parar --> Novo    --> Manual  --> Repetir --> Auto    --> Sair
Parar --> Repetir --> Auto    --> Novo    --> Manual  --> Sair
Parar --> Auto    --> Repetir --> Novo    --> Manual  --> Sair
Parar --> Novo    --> Auto    --> Repetir --> Manual  --> Sair
Parar --> Auto    --> Novo    --> Manual  --> Repetir --> Sair
```

Total de Caminhos sem Repetições = 204

ANEXO III

Conteúdo do disquete:

- uma cópia da versão inicial de ADTTool;
- alguns exemplos de programas *cpp* para teste;

Arquivos .cpp de Teste para a ferramenta ADTTool

inverte lista.cpp: inverte os elementos de uma lista linear;

inverte.cpp: inverte os elementos de uma lista linear usando uma pilha como auxiliar;

ordena e cria.cpp: dadas suas listas lineares; ordenar as listas se estiverem desordenadas e criar uma terceira lista com os elementos da primeira sem os elementos que pertencem à interseção da primeira com a segunda;

ordena lista.cpp: ordenar os elementos de uma lista linear de números inteiros quaisquer;

ordena1 pilha.cpp: ordenar os elementos de uma pilha de números inteiros quaisquer (crescente da base para o topo), usando apenas uma pilha como auxiliar

ordena2 pilha.cpp: ordenar os elementos de uma pilha de números inteiros quaisquer (crescente da base para o topo, usando duas pilhas como auxiliares para a solução);

prog38.cpp: criar a lista linear L1 com números; criar L2 com os elementos da L1 maiores do que 1;

prog44.cpp: criar duas listas lineares:l1 e l2, e montar l3 com os elementos da interseção entre as duas listas; no final l1 e l2 deverão estar vazias;

prog47.cpp:exemplo para usar todos os espaços reservados para as estruturas;

prog50.cpp: dadas três estruturas: uma pilha, uma lista e uma fila; montar a pilha e a lista com novas informações; eliminar os elementos da lista e da pilha e inserir na fila;

prog51.cpp: montar duas lista lineares:l1 e l2 e criar l3 com os elementos de da primeira lista sem os elementos pertencentes à interseção;

prog54.cpp: teste de uma função com erro na definição e uso de variáveis; as variáveis são definidas com letra maiúscula e depois se passa a utilizar letra minúscula;

prog54a.cpp: função anterior com os erros corrigidos;

prog60.cpp: dada uma lista linear; empilhar apenas as informações pares da lista linear;

sem interseção.cpp: dadas duas listas lineares ordenadas; criar uma terceira lista com os elementos da primeira sem os elementos que pertencem à interseção da primeira com a segunda;

troca.cpp: troca os elementos de uma pilha para outra