

Adriano Sunao Nakamura

ANEXO AO PROCESSO Nº 243/97 - CM

DEFESA DE TESE DE DOUTORADO EM: 11/12/2001.

PROGRAMA DE PÓS-GRADUAÇÃO EM EDUCAÇÃO

Um Estudo Sobre o Ensino da Lógica de Programação
nos Cursos Superiores de Informática

Marília
2001



Adriano Sunao Nakamura

**Um Estudo Sobre o Ensino da Lógica de
Programação nos Cursos Superiores de
Informática**

**Faculdade de Filosofia e Ciências
Universidade Estadual Paulista
Marília – 2001**



Nº cham. 371.334
N 142u

Nº Tombo 1295

Aquisição D

Preço 5,00

Data 10 / 4 / 02

Proced. autor

NAKAMURA, Adriano Sunao

Um estudo sobre o ensino da lógica de programação nos cursos superiores de informática / Adriano Sunao Nakamura – Marília, 2001

163 p.: 30 cm

Tese (Doutorado) – Faculdade de Filosofia e Ciências – Universidade Estadual Paulista, 2001.

Bibliografia: p. 115-122

1. Informática Educativa. 2. Educação em Informática. 3. Visualização de Software. I. Autor. II. Título.

CDD20 – 371.334

Adriano Sunao Nakamura

**Um Estudo Sobre o Ensino da Lógica de
Programação nos Cursos Superiores de
Informática**



**Tese apresentada à Universidade
Estadual Paulista – Câmpus de Marília,
como exigência parcial para a obtenção
do título de Doutor em Educação**

Orientador: Prof. Dr. Paschoal Quaglio

1410001295



**Faculdade de Filosofia e Ciências
Universidade Estadual Paulista
Marília – 2001**

Adriano Sunao Nakamura

Um Estudo Sobre o Ensino da Lógica de Programação nos Cursos Superiores de Informática

Banca Examinadora

**Faculdade de Filosofia e Ciências
Universidade Estadual Paulista
Marília - 2001**



Agradecimentos

As teorias, idéias e conclusões expressas neste trabalho demandaram muito esforço e tempo para serem concretizadas até darem origem a uma tese. No transcorrer desse período, pude contar com pessoas que trouxeram grandes contribuições.

Meus agradecimentos são expressos:

À **Faculdade de Ciências Exatas e Tecnológicas da Universidade de Marília**, pelo apoio à pesquisa, dando-me amplas condições para que a ferramenta AVA pudesse ser experimentada com êxito.

Ao **Prof. Dr. Pascoal Quaglio**, pelo voto de confiança dado a este trabalho, manifestado ao assumir a orientação às vésperas da qualificação. Apesar da breve convivência, sob sua orientação recebi a compreensão e a consideração que se pode esperar de um grande mestre.

Ao **Prof. Dr. Cosme Damião Bastos Massi**, pela orientação inicial, pela amizade e por tudo aquilo que direta e indiretamente contribuiu para estabelecer as bases deste trabalho.

Ao **Prof. Edson Francisco da Fonseca**, pela assistência com a linguagem JAVA, permitindo que a ferramenta fosse desenvolvida a contento.

Aos **Profs. Dr. Marcos Luís Mucheroni, Dr. José Carlos Miguel e Dra. Silvana Aparecida B. G. Vidotti**, pelas revisões e sugestões por ocasião da qualificação, as quais muito contribuíram para a conclusão deste trabalho.

Aos **Profs. Alexandre Marcelino da Silva, Ana Paula P. M. Peruzza, Edson Francisco da Fonseca, Ildeberto Aparecido Rodello, Janaína Cintra Abib, João Carlos de M. Morselli Jr., Júlio Fernando Lieira, Maurício Duarte, Renata Ap. de Carvalho Paschoal e Ricardo Rodrigues Ciferri**, pela inestimável contribuição que deram ao avaliar a ferramenta AVA.

Aos **professores e alunos da Faculdade de Ciências Exatas e Tecnológicas**, pela receptividade dada ao trabalho de campo, sem o qual seria impossível a sua conclusão.

Dedicatória

Dedico este trabalho
a meus pais, **Iukio Nakamura** e **Tiyako Nakamura**,
que, apesar de serem pessoas simples e humildes, investiram em mim anos e anos
de trabalho, afeto e compreensão, dando-me plenas condições para que
eu galgasse todos os obstáculos até alcançar mais este objetivo.

E a **Gislene Aparecida Dourado**, companheira e amiga de
todas as horas, a quem expresso minha profunda gratidão e amor.

Resumo

O processo de ensino-aprendizagem da lógica de programação nos cursos voltados à área de informática apresenta-se como um desafio tanto aos educadores como aos alunos; sérias conseqüências podem ser acarretadas no decorrer do curso devido ao mau desempenho na disciplina em questão. Sendo assim, o presente trabalho tem como objetivo principal realizar um estudo sobre o ensino da lógica de programação constatando a necessidade de se oferecer uma nova forma de abordar este processo com o objetivo de motivar o aluno e melhorar o seu desempenho. Este estudo inicia-se com uma revisão da literatura sobre o uso do computador na educação, apontando as profundas mudanças causadas no processo educacional, e estende-se para área de pesquisa denominada **Visualização de Software**; alguns sistemas de cunho didático que auxiliam no ensino de lógica de programação são avaliados para se estabelecer alguns requisitos básicos desta categoria de aplicação; tais requisitos são utilizados como subsídio para o desenvolvimento do sistema AVA – Ambiente para Desenvolvimento e Visualização de Algoritmos. Este estudo culmina com (1) a caracterização do ensino da lógica de programação, discutindo a ementa da disciplina, traçando o perfil dos alunos matriculados nos cursos de Ciência da Computação e Sistemas de Informação (Universidade de Marília), e analisando o seu desempenho e as dificuldades enfrentadas, e (2) a avaliação da ferramenta desenvolvida através da sua aplicação em caráter experimental na disciplina Construção de Algoritmos.

Palavras-chave: Informática Educativa, Educação em Informática, Visualização de Software, Animação de Algoritmo



Abstract

The teaching-learning process of the programming logic in the courses has turned to the computer science area becomes a challenge so much to the educators as to the students; serious consequences may be carted in elapsing of the course due to the bad acting of the discipline in subject. Therefore, the present work has as main objective to accomplish a study on the teaching of the programming logic verifying the need to offer a new form of approaching this process with the objective of to motivate the student and to improve his performance. This study has begun with a review of the literature on the use of computer in the education, causing deep changes in the educational process, and it extends for the research area named Software Visualization; some systems of didactic stamp that aid in the teaching of programming logic they are appraised to settle down some basic requirements of this application category; these requirements are used as subsidy for the development of AVA (Ambiente para Desenvolvimento e Visualização de Algoritmos) system. This study has ended with (1) the characterization of the programming logic teaching discussing the curricula of the discipline, tracing the profile of students matriculated in the courses of Computer Science and Information Systems (Universidade de Marília) and analyzing his acting and the faced difficulties and (2) the evaluation of the developed tool through its application in experimental character in the discipline Algorithms Construction.

Keywords: Educational Computer Science, Education in Computer Science, Software Visualization, Algorithm Animation

Sumário

Considerações Iniciais.....	1
Motivação para o Estudo	2
Objetivos	3
Metodologia de Pesquisa.....	4
1 O Computador na Educação.....	6
1.1 O Papel do Computador no Processo Educacional.	7
1.1.1 Posicionamento dos Educadores sobre o Uso da Tecnologia	7
1.1.2 O Computador como Máquina de Ensinar.....	9
1.1.3 Construcionismo	11
1.2 Mudança de Paradigma	13
1.2.1 O Conhecimento Visto como uma Rede	15
1.2.2 O Uso da Tecnologia na Formação dos Significados	17
1.3 O Papel do Professor Diante da Tecnologia.....	18
2 Sistemas de Visualização	20
2.1 Visualização Científica	21
2.2 Visualização de Software	23
2.2.1 Uma Definição.....	24
2.2.2 Sistemas de Visualização (Animação) de Algoritmos	27
2.2.3 Sistemas de Visualização de Programas	28
2.3 Exemplos de Sistemas de Visualização de Software	28
2.3.1 Balsa	29
2.3.2 Zeus.....	29
2.3.3 XTango	30
2.3.4 Astral	31
2.3.5 Leonardo	32

2.3.6	Alma	33
2.3.7	Polka.....	35
2.4	O Papel dos Sistemas de Visualização de Software na Educação	35
2.4.1	As Diversas Formas de Utilização	37
2.4.2	Benefícios Esperados no Processo de Ensino-Aprendizagem de Lógica de Programação	38
3	Avaliação de Algumas Ferramentas que Auxiliam o Processo de Ensino-Aprendizagem em Lógica de Programação.....	41
3.1	Estabelecimento dos Requisitos a Serem Considerados na Avaliação de Ferramentas que Auxiliam o Processo de Ensino-Aprendizagem de Lógica de Programação.....	42
3.1.1	Considerações sobre o Paradigma de Programação.....	42
3.1.2	Considerações sobre a Gramática da Linguagem Algorítmica.....	44
3.1.3	Suporte a Edição e Compilação do Algoritmo	44
3.1.4	Visualização de Código e de Dados	45
3.1.5	Controle da Velocidade de Execução	46
3.2	As Ferramentas Avaliadas	47
3.2.1	Portugol/Plus	48
3.2.2	ILA	50
3.2.3	DynaLab	51
3.2.4	PascalX.....	54
3.2.5	Construtor	57
3.3	Comparativo entre as Ferramentas Analisadas.....	58
4	AVA (Ambiente para Desenvolvimento e Visualização de Algoritmos)	61
4.1	Especificação dos Requisitos.....	62
4.2	Arquitetura do Sistema AVA	63
4.3	A Linguagem Algorítmica	64



4.4	Módulo de Edição	66
4.5	O Compilador	70
4.5.1	Analisador Léxico.....	70
4.5.2	Analisador Sintático	73
4.5.3	Analisador Semântico	76
4.5.4	Tabela de Símbolos / Estrutura Dinâmica para Representação do Algoritmo.....	77
4.6	Ambiente Visualizador	79
4.6.1	Visualização do Código	81
4.6.2	Visualização dos Dados.....	81
4.6.3	Janela de Entrada e Saída	82
4.6.4	Controle da Execução do Algoritmo	83
4.7	Características, Limitações e Deficiências da Ferramenta	84
5	Caracterização do Ensino de Lógica de Programação nos Cursos Superiores de Informática	86
5.1	O Ensino da Lógica de Programação	87
5.2	A Abordagem Dada ao Ensino de Programação nos Cursos de Algumas Instituições Brasileiras.....	89
5.3	Uma Experiência Realizada na Faculdade de Ciências Exatas e Tecnológicas da Universidade de Marília	94
5.3.1	Perfil dos Alunos dos Cursos de Ciência da Computação e Sistemas de Informação na Disciplina de Construção de Algoritmos.....	95
5.3.2	A Disciplina Construção de Algoritmos.....	98
5.3.3	A Deficiência Apresentada na Disciplina Construção de Algoritmos.....	99
5.3.4	Análise Comparativa entre as Turmas de 2000 e 2001	103
5.3.4.1	A Turma de 2000	103
5.3.4.2	A Turma de 2001	104
5.3.4.3	A Comparação.....	104
	Considerações Finais.....	111



A Experiência com o Uso da Ferramenta AVA.....	112
Reestruturação do Ambiente de Ensino-Aprendizagem	113
Bibliografia	115
Anexo I - Tabela de Tokens	123
Anexo II – Árvore de Derivação Sintática da Linguagem Algorítmica	124
Anexo III – Gramática da Linguagem Algorítmica (<i>Notação BNF - Backus-Naur Form</i>).....	128
Anexo IV – Conteúdo Programático da Disciplina Construção de Algoritmos.....	133
Anexo V – Questionário submetido aos alunos que ingressaram em 2000 e 2001 nos cursos de Bacharelado em Ciência da Computação e Sistemas de Informação / Tabulação das Respostas	138
Anexo VI – Descrição da Linguagem Algorítmica.....	142



Índice de Figuras

Figura 1.1 – Modelo instrucionista de ensino	10
Figura 1.2 – Ciclo descrição-execução-reflexão-depuração que se estabelece na atividade de programação	12
Figura 2.1 – Estágios do processo de visualização	22
Figura 2.2 – Diagrama relacionando os termos utilizados em Visualização de Software ..	26
Figura 2.3 – Tela com a animação do algoritmo do caminho mais curto (Dijkstra).....	29
Figura 2.4 – Animação do algoritmo HeapSort em três dimensões.....	30
Figura 2.5 – Animação no XTango	31
Figura 2.6 – Animação de uma árvore B - antes da inserção do elemento “g”	32
Figura 2.7 – Animação de uma árvore B - após a inserção do elemento “g”	32
Figura 2.8 – Animação no Leonardo	33
Figura 2.9 – Animação de um programa em Pascal	34
Figura 2.10 – Animação do algoritmo <i>quicksort</i> paralelo	35
Figura 3.1 – Ambiente de desenvolvimento do Portugol/Plus	49
Figura 3.2 – Processo de compilação do Portugol/Plus.....	49
Figura 3.3 – Execução de um programa desenvolvido com o ILA	50
Figura 3.4 – Módulo editor/compilador do DynaLab.....	52
Figura 3.5 – Ambiente de execução e visualização do DynaLab.....	52
Figura 3.6 – Ambiente de desenvolvimento do PascalX.....	54
Figura 3.7 – Gabarito para edição de comandos	55
Figura 3.8 – Pré-execução em modo DOS.....	56
Figura 3.9 – Ambiente de visualização da execução.....	56
Figura 3.10 – Ambiente de desenvolvimento e visualização do Construtor.....	57



Figura 3.11 – Gabarito para edição de comandos	58
Figura 4.1 – Fases do processo de visualização de um algoritmo/programa	62
Figura 4.2 – Arquitetura do sistema AVA	64
Figura 4.3 – Módulo de edição e compilação.....	67
Figura 4.4 – Menu principal do tutorial	68
Figura 4.5 – Acesso ao tutorial.....	68
Figura 4.6 – Opções do menu Gabarito	69
Figura 4.7 – Gabarito para edição de comandos	69
Figura 4.8 – Esquema de compilação de um algoritmo.....	71
Figura 4.9 – Lista encadeada armazenando os tokens reconhecidos	72
Figura 4.10 – Árvore de derivação sintática	74
Figura 4.11 – Estrutura dinâmica (parcial) que permite a execução do algoritmo	78
Figura 4.12 – Ambiente de Execução e Visualização.....	80
Figura 4.13 – Janela de leitura.....	83

Índice de Quadros

Quadro 3.1 – Comparativo entre as ferramentas.....	59
Quadro 4.1 – Estrutura do algoritmo.....	65
Quadro 4.2 – Exemplo de um programa escrito na linguagem algorítmica.....	66
Quadro 4.3 – Código inicial gerado automaticamente	67
Quadro 4.4 – Sentença de um programa fonte em linguagem algorítmica	71
Quadro 4.5 – Tokens dos símbolos.....	72
Quadro 4.6 – Regras de derivação utilizadas para gerar a árvore sintática da figura 4.10...	75
Quadro 4.7 – Declaração de uma variável composta.....	81
Quadro 4.8 – Representação de um vetor na janela de visualização de dados	82
Quadro 5.1 – Comparação da disciplina entre diversas instituições tradicionais	92
Quadro 5.2 – Ementário da disciplina Construção de Algoritmos	98
Quadro 5.3 – Critério de avaliação	101



Índice de Gráficos

Gráfico 5.1 – Índices de aprovação, reprovação e desistência.....	101
Gráfico 5.2 – Valor médio das notas nas disciplinas.....	101
Gráfico 5.3 – Comparação entre as turmas 2000 e 2001.....	105
Gráfico 5.4 – Comparação entre as médias das notas da turma de 2000 e 2001	106
Gráfico 5.5 – Comparação entre as idades médias.....	107
Gráfico 5.6 – Porcentagem de alunos que trabalham	107
Gráfico 5.7 – Porcentagem de alunos que arcam com o custeio do estudo.....	107
Gráfico 5.8 – Índices de aprovação das turmas de SI.....	108
Gráfico 5.9 – Valor médio das notas das turmas de SI.....	108
Gráfico 5.10 – Evolução do conteúdo programático.....	109

Considerações Iniciais

Os computadores são máquinas “incríveis”. Eles parecem ser capazes de realizar qualquer tipo de tarefa valendo-se de um conjunto pequeno de operações (tudo o que eles fazem é manipular *zeros* e *uns* numa velocidade extremamente rápida). Essas máquinas auxiliam médicos em complicados procedimentos cirúrgicos, ajudam cientistas a realizarem sofisticados cálculos matemáticos, fornecem com precisão a previsão meteorológica com dias de antecedência... Tudo isso dá a entender a uma pessoa leiga que se tratam de seres inteligentes. No entanto, por trás dessa poderosa máquina encontra-se o “homem”, que é o “ser” responsável por instruí-la para realizar todas essas tarefas.

Apesar da simplicidade apresentada por suas operações (inverter, zerar e comparar *bits*), o ato de programar um computador pode ser considerado uma atividade cognitiva extremamente complexa e envolve um raciocínio bastante apurado do programador na tarefa de encontrar a solução adequada a um determinado problema e implementá-la de forma eficiente. Nesta tarefa, dificilmente o primeiro esboço de um programa está livre de erros e, conseqüentemente, grande parte do tempo de seu desenvolvimento é devotada a testes e correções e, mesmo assim, seu funcionamento inadequado pode acarretar graves conseqüências. Tais fatos vêm salientar a importância de se aprender a programar correta e eficientemente.

Considerando-se a forma como a programação é ensinada, seu princípio fundamental está relacionado à lógica de programação, mais precisamente à elaboração, compreensão e análise de algoritmos (KEHOE; STASKO, 1996).

Já é fato notório aos educadores que atuam no ensino em cursos de Ciência da Computação e afins a dificuldade enfrentada pelos estudantes neste tópico. Esta deficiência vem acarretar sérios problemas no decorrer do curso, uma vez que a lógica de

programação serve como base para todas as outras disciplinas que envolvem o uso de uma linguagem de programação.

Ford (1993) coloca três situações inconvenientes quando algoritmos e linguagens de programação são ensinados, aprendidos e estudados empiricamente:

- no ensino, é difícil projetar idéias envolvendo movimento - por exemplo, expor o funcionamento de um construtor de repetição - usando materiais de ensino convencionais (giz e quadro negro);
- quando programas são executados, caso não haja saídas visualmente informativas, os estudantes são privados de situações em que poderiam checar a compreensão tanto da linguagem que estão utilizando como da lógica embutida em seus programas;
- de forma similar, os avaliadores dos programas desenvolvidos pelos estudantes também são privados de uma dimensão visual e precisam recorrer à elaboração de situações de teste para avaliar a solução ou, então, examinar o texto do programa fonte detalhadamente.

Assumindo-se a importância que o estudo de algoritmo ostenta num curso voltado à área de informática, torna-se necessário utilizar recursos eficazes na transmissão desse conhecimento ao mesmo tempo que sejam atrativos e motivadores aos estudantes.

Motivação para o Estudo

O ensino de ciências, de uma forma geral, é relativamente direto: um conceito é introduzido e, a partir daí, um exemplo ou um experimento é apresentado para demonstrar o conceito na prática. Por exemplo, um instrutor de física explica o conceito de velocidade angular e realiza um experimento utilizando discos com engrenagens para demonstrá-lo aos alunos. Técnicas similares podem ser aplicadas a ciências como matemática, engenharia e química, entre outras. No ensino de ciência da computação, estas técnicas de experimentação/demonstração ainda são pouco utilizadas e, segundo Boroni et al. (1996), tal fato pode vir a ser um fator de desmotivação do aluno.

“Como nós, educadores, podemos atrair jovens estudantes para a ciência da computação? Como podemos transmitir a emoção de nossa disciplina a estudantes que

podem ter somente um interesse passageiro em programação e computação, encorajando-os a continuar com os estudos?” - estas são algumas das indagações feitas por Boroni et al. (1996), demonstrando sua preocupação com jovens estudantes que abandonam seus estudos logo após o início do curso, por sentirem-se desmotivados.

Segundo os mesmos autores, - em resposta às suas indagações - para tentar evitar a evasão no curso de Ciência da Computação e manter os estudantes motivados, as bases de suas disciplinas devem ser tratadas de uma forma clara, sem superestimar ou subestimar o conhecimento e as experiências anteriores do estudante.

Como *base das disciplinas de computação* encontra-se um termo que expressa muito bem a sua essência: **algoritmos** (HAREL, 1992) – “uma seqüência finita de instruções bem definidas e não ambíguas, onde cada uma pode ser executada mecanicamente num período de tempo finito com quantidade de esforço finita”. Um algoritmo sempre está associado a um dado objetivo, ou seja, à solução de um problema. A execução de um algoritmo garante que o seu objetivo seja atingido. Uma vez elaborado corretamente, o problema será solucionado de forma eficiente. Daí a preocupação dos educadores em fazer com que os estudantes consigam assimilar a sua essência.

Nos últimos anos, esforços consideráveis têm sido despendidos no sentido de proporcionar ao estudante uma forma mais eficiente de compreender o funcionamento dos algoritmos. A maior parte destes esforços tem se voltado à utilização de recursos computacionais para a exibição da dinâmica da execução de um programa (PAXTON et al., 1994).

Pesquisadores (KEHOE; STASKO, 1996; BYRNE et al., 1996; LAWRENCE et al., 1994; PRICE et al., 1993, 1998; entre outros) têm utilizado termos como **Visualização de Software** e **Animação de Algoritmos** para designar a área de pesquisa responsável por elaborar ferramentas computadorizadas que auxiliem no ensino de algoritmos e linguagens de programação, tendo como princípio básico a exibição do seu funcionamento.

Objetivos

O objetivo fundamental deste trabalho é avaliar a viabilidade de uma abordagem alternativa dada ao ensino de algoritmos – em cursos superiores da área de informática –



realizada através do uso de uma ferramenta de visualização de software, e compreende as seguintes atividades:

- Realização de um estudo detalhado sobre a área de pesquisa denominada **Visualização de Software**.
- Análise de algumas ferramentas para auxílio ao ensino de lógica de programação e estabelecimento dos requisitos básicos desta categoria de aplicação.
- Relato do processo de desenvolvimento de uma ferramenta para auxiliar o ensino de lógica de programação
- Caracterização do ensino da lógica de programação (através da disciplina Construção de Algoritmos) nos cursos de Ciência da Computação e Sistemas de Informação da Universidade de Marília, discutindo a ementa da disciplina, traçando o perfil dos alunos matriculados e analisando o seu desempenho e as dificuldades enfrentadas com a aplicação da ferramenta desenvolvida na disciplina em questão.

Metodologia de Pesquisa

Em função do objetivo estabelecido, o estudo desenvolvido divide-se em quatro momentos. No primeiro, que compreende os capítulos 1 e 2, opta-se pela metodologia da leitura interpretativa. No capítulo 1 é apresentada uma discussão sobre o uso do computador na educação, abordando suas diversas formas de uso e as profundas mudanças causadas no processo educacional. Já no capítulo 2 são discutidos os fundamentos da área de pesquisa denominada **Visualização de Software**; os tópicos abordados abrangem conceitos e histórico, exemplos de aplicações, estudos de sua aplicabilidade e benefícios esperados no processo educacional.

O segundo momento (capítulo 3) é constituído por uma pesquisa empírica que compreende a obtenção e a análise de algumas ferramentas que auxiliam no processo de ensino de lógica de programação, onde é discutida sua funcionalidade e são destacadas suas potencialidades e deficiências. Esta discussão tem a finalidade de estabelecer os requisitos a serem considerados na implementação e na avaliação desta categoria de ferramenta.



O terceiro momento (capítulo 4) detém-se no processo de implementação de uma ferramenta para auxiliar o ensino de lógica de programação. Neste processo, é adotada uma metodologia genérica para o desenvolvimento de sistemas abrangendo atividades como a análise dos requisitos, a especificação da ferramenta, a definição das estruturas de dados e da interface com o usuário, além da criação de uma pseudolinguagem utilizando termos em português.

O quarto momento (capítulo 5), abrangendo a caracterização do ensino de lógica de programação nos cursos de Ciência da Computação e Sistemas Informação da Universidade de Marília, é realizado mediante:

- leitura interpretativa das Diretrizes Curriculares dos Cursos da Área de Computação e Informática estabelecidas pelo Ministério da Educação – MEC (1998) e do Currículo de Referência recomendado pela Sociedade Brasileira de Computação – SBC (1999), entre outras referências;
- elaboração de um quadro comparativo sobre a disciplina que envolve o ensino da lógica de programação em algumas instituições de ensino tradicionais na área de computação;
- aplicação de um questionário aos alunos;
- entrevistas e conversas informais com os alunos;
- exame de registros (notas obtidas pelos alunos na disciplina);
- estudo, através de experimentação da ferramenta implementada, para avaliar a sua efetividade no processo de ensino-aprendizagem de lógica de programação.



Capítulo 1

O Computador na Educação

Motivada pelo desenvolvimento dos recursos tecnológicos, a discussão da utilização do computador no processo educacional tem ocupado lugar de destaque nas preocupações de professores e de especialistas em tecnologia educacional.

Indubitavelmente, o emprego da tecnologia de computadores na educação abriu amplas possibilidades para se atingir melhores resultados na qualidade de ensino, no entanto, por si só, isto não é um fato consumado. O que se pode notar atualmente, é que o computador causa certo fascínio no âmbito educacional: *a nova tecnologia resolverá todos os problemas de cunho educacional e transformará a sala de aula tradicional em poderosa área de aprendizagem*. Esta é uma visão mítica que parte dos educadores, principalmente aqueles que desconhecem as reais potencialidades dos computadores, tem sobre o uso da tecnologia.

Uma visão mais realista do uso da tecnologia no processo educacional estabelece que o computador nada mais é que uma mera ferramenta. Ou seja, a qualidade da aprendizagem produzida pela tecnologia depende da qualidade das considerações educacionais sobre o seu uso e não da sofisticação alcançada pela própria tecnologia. Valente (1993b) coloca que “... não se deve apenas *automatizar* os princípios educacionais e depois acreditar que se tem renovado o setor educacional. Trata-se de um sério problema no qual o uso de computadores é muitas vezes feito sem relação com suficientes princípios teóricos”.

Neste capítulo, é apresentada uma discussão sobre o uso do computador no processo educacional, destacando suas diversas formas de uso, as profundas mudanças ocasionadas no processo de ensino-aprendizagem, e o papel do professor diante da tecnologia.



1.1 O Papel do Computador no Processo Educacional.

É inegável a importância do computador na educação como instrumento de complementação ao ensino, propiciando ao processo pedagógico a obtenção de resultados mais promissores e dando oportunidade ao aluno de entrar em contato com a informática na mais tenra idade.

A introdução do computador nas escolas e nos lares está provocando uma drástica revolução na educação e nas relações entre pais, alunos e professores. A multimídia, os programas que mesclam jogos e informações educativas, os programas educativos e outras obras de referência em mídia digital, com destaque especial a Internet, vêm revolucionando o processo de aprendizagem. Na escola, o emprego destas tecnologias determina o inter-relacionamento de todas as disciplinas, e o professor, a partir deste momento, transforma-se numa espécie de condutor/facilitador, estimulando seus alunos a navegarem pelo conhecimento e a fazerem suas próprias descobertas.

Neste cenário, já não cabe mais a concepção arcaica de uma sala de aula composta por fileiras de estudantes, sentados lado a lado, encarando bem em frente o professor, que é o detentor e o fornecedor do conhecimento. A era da informação de hoje necessita de um novo modelo para a educação, por isso existe o potencial para profundas mudanças no aprendizado.

1.1.1 Posicionamento dos Educadores sobre o Uso da Tecnologia

O computador pode provocar mudanças drásticas no processo de ensino-aprendizagem, justamente pelo fato de ele ser capaz de ensinar. Entretanto, o que se deixa transparecer, é que o emprego dos computadores na educação tem criado mais controvérsias e confusões do que auxiliado na resolução dos problemas da educação. Segundo Valente (1993a), o uso do computador na educação, além de provocar o questionamento dos métodos e da prática educacional, também causou insegurança em alguns professores menos informados que receiam e refutam o uso do computador na sala de aula. Entre outros fatores, esses professores pensam que serão substituídos pela máquina. Outros já vêem o computador como uma solução para todos os males da educação.



Diante deste contexto, o posicionamento sobre o emprego de computadores na educação pode se dar sob três situações claramente definidas: entre o ceticismo e o otimismo uma grande parcela dos educadores tratam-no com indiferença.

Segundo Larsen (1999), a visão ceticista

... é geralmente caracterizada por um grande conhecimento da Educação e pouco conhecimento sobre computadores e tecnologia, o que leva à crença de que a nova tecnologia irá destruir toda a Educação humana e, como consequência derradeira, matar a própria cultura...

Rotulados de *apocalípticos* por Bianchetti (1997), os céticos acreditam que o professor poderá ser substituído pelo computador e, como consequência da interação entre a criança e uma máquina “desumana”, propiciará a formação de indivíduos desumanos e robóticos.

Ainda sob o rótulo dos céticos, Valente (1993b) aponta aqueles que exibem argumentos quanto à carência material do sistema educacional vigente: “...a escola não tem carteiras, não tem giz, não tem merenda e o professor ganha uma miséria. Nessa pobreza, como falar em computador?...”.

Já sob o rótulo dos otimistas, essa visão é caracterizada por um grande conhecimento de computadores e da tecnologia, e pouco conhecimento sobre a Educação (LARSEN, 1999). Esta leva a uma crença de que, para ensinar e educar, é necessário apenas embutir todo o conhecimento em programas de computador e apresentá-lo aos alunos. De acordo com Valente (1993b), esta é apresentada por

... todos aqueles que de certa forma acabam por abstrair problemas bem concretos da cotidianidade de nossas escolas, achando que um voluntarismo puro e simples pode resolver todas as situações que se apresentem desfavoráveis a tal empreendimento. Acreditam na potencialidade da tecnologia como resposta às necessidades do sistema educacional, sem, contudo, perceber a importância de uma maior compreensão dos efeitos e das condições de contorno do uso intensivo desses recursos...

A despeito de toda a discussão entre céticos e otimistas, segundo a perspectiva de Bianchetti (1997), aqueles que assumem uma posição de indiferença são os que causam maior preocupação. Tratado na marginalidade da discussão, Bianchetti coloca que “... a

indiferença é uma das atitudes mais perniciosas, pois elimina qualquer possibilidade de ação e de diálogo, na medida em que se nega ao enfrentamento do debate, deixando que um *laissez-faire* tome conta da situação...”. A posição dos indiferentes é de profundo desinteresse: eles ficam na expectativa sobre o rumo para o qual a discussão tende a tomar para depois assumirem uma postura mais decisiva.

Diante desse posicionamento, Bianchetti defende a instituição de uma quarta visão, síntese entre as visões apresentadas, considerando a tecnologia como “... criação humana, carregada de ideologias...”. Esta visão caracteriza-se, sobretudo, pela existência de uma postura crítica diante da tecnologia, procurando compreender em que contexto histórico, sob que condições econômicas e políticas ocorrem sua absorção pelo sistema educacional, sem, contudo, deixar de incorporá-la às práticas de ensino-aprendizagem, proporcionando, com isso, uma maior democratização da sua utilização.

1.1.2 O Computador como Máquina de Ensinar

O uso do computador na educação abre possibilidades para uma ampla gama de aplicações. Na forma mais direta e fácil e, que, conseqüentemente, exige menos investimento na formação do professor, o computador assume o papel de “máquina de ensinar”. Este tipo de abordagem pedagógica é denominado *instrução auxiliada por computador* ou, simplesmente, *instrucionismo*.

Basicamente, esta abordagem consiste na automatização dos métodos tradicionais de ensino. Ou seja, o computador é utilizado como um mero transmissor de informação para o aluno. De acordo com Valente (1993b), esta concepção de educação é baseada no modelo empirista e assume que a retenção de conhecimento se dá como consequência da contigüidade e da frequência com que ele é transmitido. Se o professor dedicar-se à preparação e à transmissão do conhecimento ao aluno, e este realizar um bom trabalho na memorização desse conhecimento, estará garantido o sucesso do processo de ensino.

O instrucionismo é caracterizado pelo uso de softwares baseado em tutoriais, softwares de exercício-e-prática e jogos. Os tutoriais enfatizam a apresentação das lições ou a explicitação da informação. No exercício-e-prática a ênfase está no processo de ensino baseado na realização de exercícios com grau de dificuldade variado em que o

aluno responde às questões colocadas pelo computador (veja figura 1.1). Nos jogos educacionais a abordagem pedagógica utilizada é a exploração livre e o lúdico ao invés da instrução explícita e direta (VALENTE, 1993a). Esses softwares podem ser incrementados com características de inteligência (como os “intelligent tutorial systems”), capazes de identificar os erros mais frequentes e ajudar os alunos a superá-los, auxiliar a resolução de problemas específicos (como os sistemas especialistas), ou auxiliar o professor a planejar suas aulas ou a monitorar o desempenho dos alunos (WENGER apud VALENTE, 1993a).

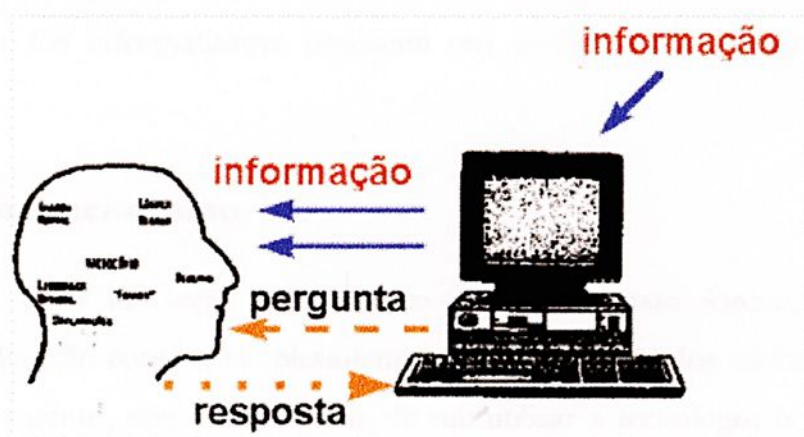


Figura 1.1 – Modelo instrucionista de ensino (VALENTE, 1993b)¹

De acordo com Valente (1993a), a considerar o processo de ensino baseado simplesmente na transmissão de informação, os programas de computador que promovem o ensino (softwares baseados em CAI – *Computer-Aided Instruction*) podem substituir o papel do professor:

- o computador tem maior capacidade para reter a informação e ministrá-la de uma maneira sistemática, meticulosa e completa (jamais se esquece de um detalhe, se isso estiver especificado no seu programa, e uma dor de cabeça ou um problema familiar jamais altera a sua performance);
- essa capacidade de sistematização do computador permite o acompanhamento do aluno em relação aos erros mais frequentes e à ordem de execução das tarefas (freqüentemente o professor tem muita dificuldade em realizar esse acompanhamento que pode ser feito pelo computador de uma maneira muito mais detalhada);

¹ Fonte: <http://www.divertire.com.br/artigos/img/valente2-1.gif>

- os sistemas computacionais apresentam hoje diversos recursos de multimídia, como cores, animação e som, possibilitando a apresentação da informação de um modo que jamais o professor poderá fazer com giz e quadro negro, mesmo que ele use giz colorido e seja um exímio comunicador.

Apesar de toda essa “aparente” eficiência da informatização do ensino, os resultados em termos da adequação dessa abordagem no preparo de cidadãos capazes de enfrentar as mudanças que a sociedade está passando são questionáveis. Tanto o ensino tradicional quanto sua informatização preparam um profissional obsoleto (VALENTE, 1993b).

1.1.3 Construcionismo

Como foi visto na seção anterior, no paradigma instrucionista, o uso do computador na educação consiste simplesmente na informatização dos meios tradicionais de instrução. No entanto, esta é uma forma de sub-utilizar a tecnologia; o computador pode ser utilizado para enriquecer ambientes de aprendizagem onde o aluno, interagindo com os objetos desse ambiente, tem chance de construir o seu conhecimento.

Quando o aluno “ensina” o computador, este desempenha o papel de uma “máquina para ser ensinada”, proporcionando condições para o aluno construir o seu conhecimento. Esta abordagem pedagógica foi denominada por Papert (1994) de *construcionismo*.

De acordo com Valente (1993b), há uma confusão entre o construcionismo e o construtivismo de Piaget². Enquanto o construtivismo vê o conhecimento ser internamente construído social e culturalmente intermediado, o construcionismo preconiza que a aprendizagem deve se dar através do engajamento do aprendiz na construção de um produto significativo e, de preferência, contextualizado: (1) o aprendiz constrói alguma coisa - o aprendizado ocorre através do *fazer*, do “colocar a mão na massa”;

² O construtivismo prevê a educação a partir de uma linguagem (“leitura do mundo”) e de conceitos trazidos pelo indivíduo para a escola. O educador deve considerar o que é sabido pelo educando e, assim, educa-lo considerando a sua linguagem original (LEITE; SILVA, 2001).

(2) o aprendiz sente-se motivado por estar construindo algo de seu interesse – o envolvimento afetivo torna a aprendizagem mais significativa.

A despeito dessas colocações, na opinião de Valente (1993b), o que contribui para a diferença entre essas duas maneiras de construir o conhecimento é a presença do computador.

O uso dos computadores sob a ótica construcionista parte para uma direção oposta à do instrucionismo. Nela, o aluno, fazendo uso de um software apropriado, aprende exercitando uma tarefa cujo objetivo é “ensinar” o computador. Este software pode ser de uso geral, como linguagens de programação (BASIC, Pascal, Logo, entre outros), linguagens para criação de banco de dados ou mesmo processadores de texto. Em todos esses casos, o aluno pode representar suas idéias por meio desse software, ou seja, o computador pode ser visto como uma ferramenta que permite ao aluno resolver problemas ou realizar tarefas como desenhar, escrever etc.

Para ensinar o computador, o aluno deve utilizar conteúdos e estratégias na resolução de problemas. Valente (1993b) estabelece o ciclo **descrição-execução-reflexão-depuração** como uma estratégia para programar o computador (veja figura 1.2).

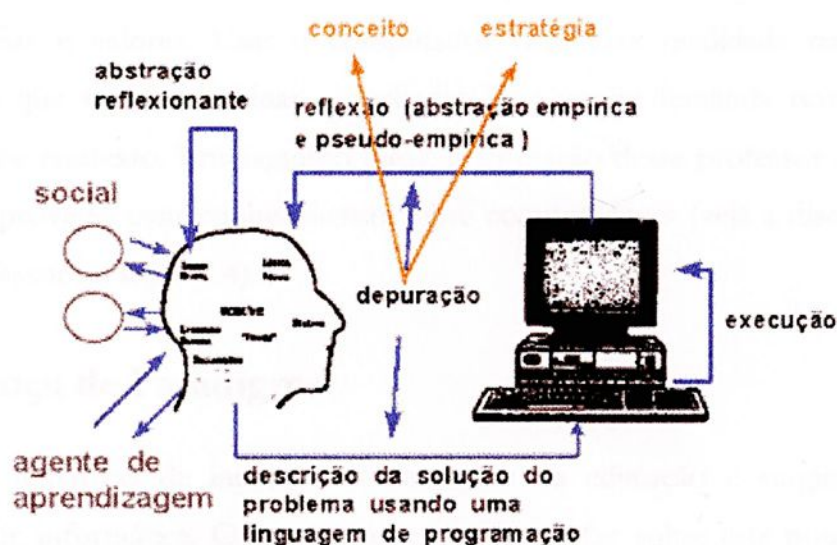


Figura 1.2 – Ciclo **descrição-execução-reflexão-depuração** que se estabelece durante a atividade de programação (VALENTE, 1993b)³

³ Fonte: <http://www.divertire.com.br/artigos/img/valente2-2.gif>

Inicialmente, a interação com o computador durante a programação requer a **descrição** de uma idéia em termos de uma linguagem formal e precisa. Esta descrição permite ao aluno representar e explicitar o nível de compreensão que possui sobre os diferentes aspectos envolvidos na resolução do problema. Na sequência, o computador **executa** fielmente a descrição, fornecendo uma resposta imediata e desprovida de qualquer animosidade ou afetividade que possa haver entre o aluno e o computador. O resultado obtido é fruto somente do que foi solicitado à máquina. A seguir, o resultado obtido serve como objeto de **reflexão** sobre o que foi solicitado ao computador. Finalmente, se o resultado não corresponde ao esperado, o aluno precisa **depurar** a idéia original, visando otimizá-la, adquirindo novos conteúdos ou estratégias.

Este ciclo não só contribui para a resolução do problema, como também oferece condições para o aluno construir novos conhecimentos. Entretanto, este ciclo não acontece simplesmente colocando o aluno frente ao computador. Segundo Valente (1993b), o uso do computador na criação de ambientes de aprendizagem que enfatizam a construção do conhecimento apresenta enormes desafios. O primeiro implica em entender o computador como uma nova maneira de representar o conhecimento, provocando um redimensionamento dos conceitos já conhecidos e possibilitando a busca e a compreensão de novas idéias e valores. Usar o computador com essa finalidade requer a análise cuidadosa do que significa ensinar e aprender bem como demanda rever o papel do professor nesse contexto. Em segundo lugar, a formação desse professor envolve muito mais do que provê-lo com conhecimento sobre computadores (veja a discussão sobre o papel do professor na seção 1.4).

1.2 Mudança de Paradigma

Várias tentativas de inovação foram feitas na educação e surgiram, assim, os laboratórios de informática. O questionamento que se faz sobre este ponto é se houve uma real inovação ou se o computador veio apenas mascarar um modelo tradicional tornando-se uma perigosa resistência à mudança. Pappert (1994) faz uma crítica aos laboratórios de informática, que em sua própria estrutura física é colocado como um corpo estranho dentro da instituição: “... geralmente são separados do restante das salas de

aula, como um organismo doente em um corpo são, que deve ser isolado...”. Nesses locais, quando as crianças não estão seguindo um currículo voltado para a informática, estão presas ao currículo vigente, não possibilitando uma real “... exploração viva e empolgante por parte dos alunos. O computador foi agora usado para reforçar os meios da escola. O que começava como um instrumento subversivo de mudança foi neutralizado pelo sistema e convertido em instrumento de consolidação...” (PAPPERT, 1994, p. 41).

A realidade do ensino atual ainda utiliza-se de uma prática pedagógica na qual se expõe uma quantidade imensa de informações (seguindo o conteúdo programático) que deve ser memorizada e após a avaliação é descartada.

Esta abordagem não leva em consideração as concepções espontâneas que os alunos trazem do seu cotidiano e, muitas vezes, é responsável pela sua desmotivação e, conseqüentemente, pela sua insuficiência na aprendizagem.

Contrapondo-se a essa situação, o *construtivismo* foi um grande avanço no sentido de evidenciar que só se aprende aquilo com o qual se entra em contato e de cuja construção se participa. No entanto, estudos recentes estão começando a mostrar que a aprendizagem se dá não somente através das funções intelectuais, mas também envolve as funções afetivas (GOUVÊIA, 1999).

Nessa direção, alguns autores vêm trazendo importantes contribuições no sentido de uma articulação entre objetividade e subjetividade, para que ocorra a aprendizagem. Junto ao conhecimento prévio, existem outros aspectos ou processos psicológicos que agem como mediadores entre o ensino e os resultados da aprendizagem; esta subjetividade está relacionada a “... a percepção que o aluno tem da escola, do professor e de suas atuações; suas expectativas perante o ensino; suas motivações, crenças, atitudes e atribuições; as estratégias de aprendizagem que é capaz de utilizar etc...” (GOUVÊIA, 1999).

Para que a aprendizagem ocorra, ela deve se dar de forma significativa, o que exige que esta seja vista como uma compreensão de significados, relacionando-se não somente às experiências anteriores dos alunos, mas também às suas vivências, permitindo a formulação de problemas de algum modo desafiadores que os incentivem a aprender mais de forma a estabelecer diferentes tipos de relações entre fatos, objetos, acontecimentos,

noções e conceitos, desencadeando modificações de comportamentos e contribuindo para a utilização do que é aprendido em diferentes situações (MACHADO, 1995).

1.2.1 O Conhecimento Visto como uma Rede

A concepção de que o processo de ensino-aprendizagem deve se dar de forma linear e paulatinamente prevaleceu na escola durante séculos. O pressuposto adotado para fundamentar essa concepção é de que existem noções e conceitos que devem necessariamente anteceder a outros (GOUVÊIA, 1999). Tal fato é utilizado para justificar o grande número de retenções nas séries escolares. No entanto, mesmo que se admita a existência de etapas a serem cumpridas ao longo da escolarização, as limitações de uma concepção linear do processo de aquisição do conhecimento evidenciam suas insuficiências para condicionar fortemente os currículos e programas.

Baseado na abordagem de que a aprendizagem deve ocorrer de forma significativa, uma nova concepção de aquisição de conhecimento tem recebido destaque na literatura. A aprendizagem significativa, vista como a criação de relações entre significados, estabelece o conhecimento como sendo uma rede de significados em permanente processo de transformação, contrapondo aos preceitos de conceito, linearidade, hierarquia e encadeamento que, até então, predominam no processo de ensino-aprendizagem, conforme coloca Machado (1995):

Neste ponto, não se pode perder de vista que a metáfora de rede contrapõe-se diretamente à idéia de cadeia, de encadeamento lógico, de ordenação necessária, de linearidade na construção do conhecimento, com as correspondentes determinações pedagógicas relacionadas com os pré-requisitos, as seriações, os planejamentos e avaliações. Em razão desse fato, convém ressaltar-se ... a pressuposição da não existência de um percurso necessário do ponto de vista lógico para se percorrer a rede, de nó em nó; nenhum nó é privilegiado nem univocamente subordinado a um outro, sendo sempre factíveis diversos percursos alternativos para os trajetos entre dois nós... (p.140).

A aprendizagem significativa de um conteúdo qualquer implica inevitavelmente em sua memorização compreensiva e armazenamento numa rede de significados. Sendo assim, na medida em que contribui para ampliar e estender tal rede de significados,

incrementa-se a capacidade do aluno para estabelecer novas relações quando se defrontar com tarefas ou situações posteriores, pelo que uma aprendizagem realizada de forma significativa é, ao mesmo tempo, uma aprendizagem que tem um elevado valor funcional, isto é, uma aprendizagem útil, uma aprendizagem que pode ser utilizada com relativa facilidade para gerar novos significados.

A construção do conhecimento dissociada do aspecto da subjetividade, da história individual e coletiva, e dos saberes populares, não assegura uma formação cultural politizada, visando preparar o indivíduo para o mundo, pois o que se procura integrar é apenas os conteúdos sistematizados e disciplinados, mantendo o seu caráter exclusivista em relação aos outros saberes (GOUVÊIA, 1999).

Longe de querer condenar ou banir as “disciplinas”, o paradigma de rede reivindica a sua integração com os saberes não disciplinados, de modo que as primeiras “ganhem vida” à luz da dinâmica das transformações da realidade (SANTIAGO, 1997). Diante deste fato, torna-se oportuno colocar as palavras de Machado (1995):

No que tange às disciplinas, a metáfora de rede nem de longe tende a mitigar sua importância. De fato, nos processos cognitivos, sempre serão necessários ordenamentos, procedimentos algorítmicos, hierarquias, ainda que o conhecimento não possa ser caracterizado apenas por estes elementos constitutivos, isoladamente ou em conjunto. Analogamente, afirmar-se a flexibilidade das fronteiras disciplinares não significa que as disciplinas tornam-se dispensáveis, seguramente elas não o são. Parafraseando o poeta, são demais os perigos desta rede de significações, com sua multiplicidade de nós e de vias de interligação, sobretudo para aqueles que nela ‘navegam’ com entusiasmo e paixão; são inúmeras as possibilidades de vagar à toa, de se perder. Para enfrentar tais perigos, sempre será necessário um mapeamento que oriente e articule os caminhos a seguir, que apresente um espectro não-hierárquico e acentrado de opções. O quadro de disciplinas fornece naturalmente um tal mapeamento... (p.155).

Convém, neste momento, ressaltar novamente a importância da conexão entre a objetividade e a subjetividade no processo de comunicação, considerando a metáfora de rede. No entanto, para que o aluno possa construir seu conhecimento dentro dos limites do contexto, da produção e da recepção faz necessário o papel de um facilitador que o conduza durante a criação de relações entre significados.

1.2.2 O Uso da Tecnologia na Formação dos Significados

O conceito de conhecimento a respeito do qual convergem as teorias contemporâneas aproxima-se cada vez mais da idéia de conhecer o significado, em que a imagem de cadeias lineares é substituída pela de teias de relações. Devido a este motivo, o computador adequa-se, particularmente, ao uso educacional. É bem conhecido o papel fundamental do envolvimento pessoal do aluno no processo de aprendizagem. Quanto mais ativamente uma pessoa participar da aquisição de um conhecimento, mais ela irá integrar e reter aquilo que aprender. “Ora, a multimídia interativa, graças a sua dimensão reticular e não-linear, favorece uma atitude exploratória ou mesmo lúdica em face do material a ser assimilado. É, portanto, um instrumento bem adaptado a uma pedagogia ativa...” (GOUVÊIA, 1999).

O uso da tecnologia de computadores propicia uma eficiente estrutura de captação e armazenamento de informações com uma capacidade enorme de difusão do conhecimento dando-lhe uma nova configuração - enquanto rede de significados - que altera o antigo paradigma do conhecimento baseado em memória. Atualmente, nesta vasta imensidão de informações que a Internet proporciona, requisita-se a capacidade de selecionar e analisar a informação que pode ser acessada.

Os sistemas baseados em hipermídia, apoiados no paradigma construcionista, podem permitir que os alunos construam o seu conhecimento baseado em compreensões prévias interagindo dinamicamente com a mídia. Estes sistemas privilegiam o controle das lições pelo aprendiz (CAMPOS et al., 1998). “O sucesso da aprendizagem está no interesse, inteligência e habilidade do aprendiz em tomar decisões sobre seqüência, momento e ênfase” (EKLUND apud CAMPOS et al., 1998).

Segundo Campos e colegas, estudos vêm sendo feitos no sentido de desenvolver sistemas adaptativos baseados no gerenciamento da interface ou na relevância da informação para o usuário. Nestes sistemas, a interface é alterada com base em diversas categorias estereotipadas e o sistema tenta traçar o conhecimento do aluno e fornece conselhos individuais. Em alguns casos, isto é feito através de uma base de conhecimento onde constam características cognitivas dos usuários, preferências, tomadas de decisão e regras para escolher o formato do aconselhamento correto. O modelo do aluno pode,

então, ser usado para alterar as características da interface e mudar segundo as necessidades do aprendiz. A aprendizagem é mais efetiva quando o controle é do usuário final, quando este toma iniciativa e escolhe cada passo na sequência de eventos de aprendizagem.

A maioria das aplicações em hipermídia apresenta-se como ambientes de aprendizagem que representam a complexidade natural dos domínios e suportam construção colaborativa do conhecimento. Mais do que apresentar a informação conectada em nós, os ambientes de aprendizagem baseados em hipermídia permitem uma profunda reflexão sobre o conteúdo que está sendo trabalhado. Isto representa a progresso natural das tecnologias de aprendizagem, através do crescimento cognitivo até a aprendizagem significativa do construtivismo (CAMPOS et al., 1998).

1.3 O Papel do Professor Diante da Tecnologia

À luz desta nova abordagem dada ao processo educacional, apregoando a incorporação do computador, o aluno se torna mais ativo, crítico e curioso; ele passa a ser autor de seu processo de aprendizagem interferindo, questionando, decidindo o que fazer e administrando seu próprio tempo.

De acordo com Freire e Prado (1996), este cenário exige do aluno competências complexas – domínios de linguagens, intuição aliada ao raciocínio lógico-formal, imaginação e criatividade; ele também deve estar aberto e tolerante para as diferenças individuais e coletivas.

Para despertar no aluno essas competências, o professor deve desempenhar um papel de desafiador, estimulador e mediador; ele deve reconstruir o conhecimento junto com seus alunos e ter consciência da importância da tecnologia, tanto no uso genérico como no profissional, em suas atividades pedagógicas, como no planejamento e na execução de seu trabalho.

Sob esta ótica, segundo (CAMPOS et al., 1998), o professor torna-se um agente cultural, um pesquisador e contínuo aprendiz, que utiliza as diversas tecnologias para sua auto-capacitação:

Despido de uma concepção de autoridade detentora do saber, deve ousar andar no limite do caos, tendo consciência que também irá aprender com seus alunos. Consciente que esta relação é condição de cidadania. Se ambos fizerem isto estarão resgatando seus papéis de autores, que elaboram, escrevem, constroem e realizam a história...

Para o professor poder desempenhar os papéis de mediador e facilitador ele necessita reformular a sua formação pedagógica com o intuito de incorporar a tecnologia às suas práticas de aula de forma eficiente. Para familiarizar-se com o computador ele precisa usá-lo nas mais variadas atividades, mesmo que elas não sejam de especial significado pedagógico nem voltadas para a sala de aula. Quando os professores tiverem com o computador a intimidade que hoje têm com o livro, descobrirão ou inventarão maneiras de inseri-lo em suas rotinas de sala de aula, encontrarão formas de criar, em torno do computador, ambientes ricos em possibilidades de aprendizagem que proporcionarão aos alunos uma educação que os motivará tanto quanto hoje o fazem, por exemplo, os jogos computadorizados e os desenhos animados.



Capítulo 2

Sistemas de Visualização

Avanços na ciência e na tecnologia têm sido frequentemente caracterizados por invenções que permitem às pessoas enxergar coisas “velhas” de novas maneiras. Ao considerar o cenário educacional vigente, estas invenções têm o potencial de provocar profundas mudanças nas práticas pedagógicas.

O uso de figuras e ilustrações como auxílio educacional é uma prática largamente utilizada: livros-texto são repletos de ilustrações e instrutores frequentemente esboçam diagramas no quadro negro para auxiliar uma explicação. A visualização suportada por computador vai um passo além da ilustração. Enquanto visualizações estáticas fornecem às pessoas a essência de como algo se parece ou é constituído, animações adaptam-se melhor em explicar a dinâmica envolvendo um processo.

Considerando a visualização como área de pesquisa, esta compreende o uso de representações visuais de dados suportados por computador com o objetivo de ampliar a cognição (DEFANTI, 1999).

Através da visualização, pesquisadores que exploram disciplinas científicas podem tirar vantagem das novas tecnologias de hardware e software para expor e elucidar uma vasta quantidade de dados científicos. Uma vez que os dados são apresentados de forma ilustrada, os cientistas são capazes de utilizar sua habilidade cerebral para fazer analogias e vínculos entre a imagem que é apresentada visualmente e a idéia mental. Isto permite uma interpretação mais fácil e uma compreensão maior dos dados científicos.

De acordo com Price et al. (1992), ironicamente o campo da **Engenharia de Software** não tem visto os benefícios da revolução causada pela visualização científica. Educadores e engenheiros de software, de uma forma geral, fazem pouco uso de sistemas que permitam visualizar a grande quantidade de complexos dados que constitui um



programa de computador, o que vêm a trazer um certo grau de dificuldade na sua compreensão.

Neste capítulo, é apresentada uma breve explanação sobre a área de pesquisa denominada **Visualização Científica** que abrange uma ampla gama de aplicações dentre elas os **Sistemas de Visualização de Software** cujo assunto é tratado com maior profundidade, apresentando alguns exemplos destes sistemas e discutindo sobre a sua aplicabilidade no ensino de programação.

2.1 Visualização Científica

Basicamente, a visualização científica consiste na aplicação de métodos gráficos para aperfeiçoar a interpretação e o significado de dados científicos (SPENCE, 2001, WARE, 2000). Os dados científicos podem ser derivados através de instrumentos de medição ou obtidos como resultado de processamentos realizados por supercomputadores. Estes dados, na forma sua forma original, podem não ser úteis até que alguma informação possa ser extraída.

Sendo assim, o objetivo da visualização científica é prover conceitos, métodos e ferramentas para criar representações visuais expressivas e efetivas a partir dos dados científicos. Estas representações visuais levam a novas percepções e aperfeiçoam a compreensão dos processos físicos, dos conceitos matemáticos e de outros fenômenos quantificáveis expressos através dos dados, que na maioria das vezes são numéricos.

O caráter interdisciplinar desta área de pesquisa torna necessária a colaboração de profissionais em psicologia e fatores humanos que oferecem uma base científica para compreender a representação visual humana e estabelecer suas habilidades e limitações. De outro lado profissionais em computação gráfica contribuem com algoritmos e ferramentas para traduzir os dados numéricos em figuras. Conta-se, ainda, com a colaboração de artistas e *designers* gráficos que se preocupam com aspectos estéticos, com o intuito de aumentar o caráter interpretativo das representações visuais.

Considerações epistemológicas acerca do processo em questão estabelecem que os dados que derivam de modelos teóricos formulados e que geram uma imagem através de simulações numéricas, devem oferecer uma visualização concebida como correta e



confiável. Possivelmente, se os modelos teóricos tomados tivessem sido outros, a imagem resultante apresentaria outras características. Frequentemente estas imagens são apresentadas para ratificar a teoria, raramente expondo a ciência com suas controvérsias e inconsistências, tendo a finalidade de promover um contato visual imediato de conceitos científicos. Estas imagens podem ser chamadas de ícones por carregar uma multiplicidade de significados explícitos e implícitos a partir da ciência, já incorporando outras crenças que são verificadas por associação com a ciência (MELEIRO, 1997).

De acordo com Ware (2000), o processo da visualização científica compreende essencialmente três estágios: (1) o mapeamento de um fenômeno real para dados científicos (números); (2) a conversão destes números em figuras e animações e (3) a interpretação das figuras pelo observador (veja figura).

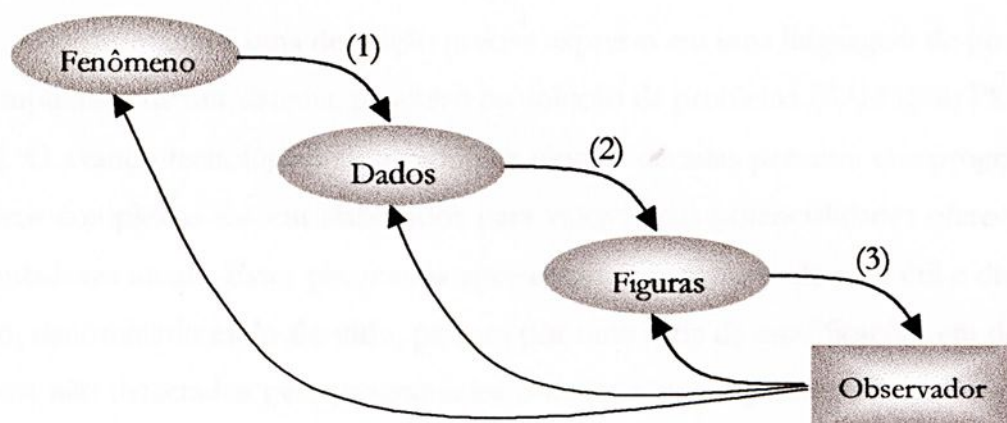


Figura 2.1 – Estágios do processo de visualização

O observador ao olhar as figuras utiliza-as como uma forma de conduzir seu pensamento, ou seja, ele tenta interpretar o significado dos números (ou o próprio fenômeno do mundo real) expresso nas figuras.

As figuras ativam o processo mental, propiciando a percepção de relacionamentos espaciais, a descoberta de padrões ou anomalias em um grande conjunto de dados, ou a compreensão intuitiva de processos complexos. Estes processos mentais obviamente são diferentes daqueles ativados se o observador estivesse interpretando os números sem o auxílio das figuras. Daí a importância de se estabelecer estratégias para se criar figuras expressivas e efetivas para garantir uma interpretação correta e significativa do observador.

O processo de interpretação dos dados inicia-se através da abstração dos dados (números) na forma de figuras. Se o mapeamento dos números em figuras não é realizado cuidadosamente, estas podem não expressar o seu verdadeiro significado e, portanto, podem levar a interpretações errôneas dos fatos científicos. O uso da cor é exemplo de um recurso para codificar valores de dados: mudanças na tonalidade podem representar valores numéricos continuamente crescentes. Uma figura é considerada expressiva se ela expressa as características fundamentais dos dados, nem mais e nem menos.

O conceito de visualização pode ser aplicado para facilitar a compreensão de um programa de computador, como é visto na seção a seguir.

2.2 Visualização de Software

Um programa é uma descrição precisa expressa em uma linguagem de programação de computador de um sistema, processo ou solução de problema (BAECKER; PRICE, 1998, p. 29). O avanço tecnológico alcançado nas últimas décadas permitiu que programas cada vez mais complexos fossem elaborados para valer-se das potencialidades oferecidas pelos computadores atuais. Estes programas apresentam um período de vida útil e durante esse tempo, denominado **ciclo de vida**, passam por uma série de modificações em decorrência de erros não detectados pelo programador durante a sua implementação e do surgimento de novas necessidades do usuário. Este processo, denominado **manutenção**, pode consumir cerca de 70% de todo o esforço despendido na concepção do programa durante todo o seu ciclo de vida (PRESSMAN, 1995). Embora seja inevitável, seus custos podem ser reduzidos de forma significativa caso erros lógicos não sejam cometidos pelo projetista/programador durante a sua concepção.

Preocupados com esta situação, educadores têm recorrido a sistemas que permitem visualizar a dinâmica do processo de execução de um programa de computador. Tais sistemas objetivam fornecer visões gráficas e animadas de componentes, métodos e técnicas utilizados em algoritmos e programas de computador. O apelo gráfico tem o poder de prender a atenção do estudante, ao mesmo tempo em que facilita o processo de compreensão de um algoritmo, tornando-o um recurso altamente importante.



Normalmente os depuradores (*debuggers*) de programas que acompanham pacotes de linguagens de programação muitas vezes se prestam a este tipo de tarefa. No entanto, pecam no aspecto pedagógico na medida que apresentam dificuldades para iniciantes em programação estudarem e experimentarem seus programas. Há, também, o agravante de apresentarem detalhes técnicos que fogem ao escopo do estudo da lógica de programação.

2.2.1 Uma Definição

O dicionário Aurélio (FERREIRA, 1999) define o termo **visualização** como “... ato ou efeito de visualizar; transformação de conceitos abstratos (que não se tem ante os olhos) em imagens reais ou mentalmente visíveis...”. O objetivo da visualização é tornar visível algo que não é, para deixá-lo mais compreensível. Sendo assim, pode-se afirmar que a **visualização de software** consiste em fazer uso de recursos gráficos e animação para representar a dinâmica da execução de um programa (PRICE et al, 1993, 1998; LAWRENCE et al., 1994; KEHOE; STASKO, 1996; BYRNE et al., 1996).

Um número razoavelmente grande de pesquisas tem sido conduzido neste sentido. Vários termos como **programação visual**, **visualização de programa** e **animação de algoritmo** têm sido definidos por pesquisadores para designar o uso de várias técnicas para aperfeiçoar a compreensão humana sobre o funcionamento de programas de computadores. Por tratar-se de uma área de pesquisa recente, vários autores têm sugerido denominações e classificações diversas para os sistemas de visualização; no entanto, estas apresentam um caráter muito subjetivo e agrupam um número de características muito reduzido para descrever uma ampla e diversa área de pesquisa (PRICE et al., 1998).

Com o intuito de direcionar as pesquisas nesta área, tendo como objetivo principal estabelecer uma linguagem comum e critérios para identificar e catalogar as descobertas dos pesquisadores, Price et al. (1993, 1998) cunharam o termo **Visualização de Software** (*Software Visualization*) para denominar esta nova área de pesquisa.

Tomando por base esta definição, o objetivo principal desta área de pesquisa é determinar como problemas externos relacionam-se a estruturas e atividades cognitivas daqueles que os observam para gerar a representação de um programa que possa ser

visualizado pelo “homem”. Visualização de Software é a tentativa de representar um programa e sua execução, utilizando-se texto, gráfico e animação.

Estudos realizados por Price et al. (1993, 1998) estabelecem as diferenças de cada um dos termos supracitados e adotam o termo **visualização de software** para englobar todos os termos anteriores. Os autores definem visualização de software como sendo o uso de recursos de tipografia, projeto gráfico, animação e cinematografia, com modernas tecnologias de interação homem-computador para facilitar a compreensão humana e o uso efetivo de softwares de computador.

De uma forma geral, sistemas de visualização de softwares oferecem facilidades aos usuários para visualizarem e interagirem com as animações geradas para representar o algoritmo ou o programa e, aos programadores para desenvolverem tais animações. Para os usuários, o sistema oferece meios de controlar os dados gerados pelo algoritmo e, também, de manipular as suas visões e a sua execução. Para os programadores, o sistema pode oferecer suporte à elaboração de animação de um algoritmo de forma tão fácil como produzir um esboço textual do mesmo (BROWN, 1991).

Sistemas de visualização de software podem ser utilizados para representar um número de diferentes perspectivas e formas de informação. São essas diferenças que determinam a categorização de formas distintas de sistemas (PRICE et al., 1993, 1998). Quando o sistema é utilizado para exibir detalhes de baixo nível sobre a execução de um programa, é designado como sistema de **visualização de programas**. Alternativamente, quando descreve o comportamento de um programa num alto nível de abstração, é denominado sistema de **visualização de algoritmos**. O diagrama exibido na figura 2.2 mostra como cada um dos termos utilizados nas demais literaturas adapta-se às definições propostas pelos autores.

De acordo com Price et al. (1993, 1998), o termo **visualização de programas** tem sido utilizado para determinar o uso de várias técnicas para aperfeiçoar a compreensão humana sobre programas de computadores. Para esse fim, a visualização deve se dar sobre o código e as estruturas de dados do programa em questão, que pode se concretizar de forma estática ou dinâmica.



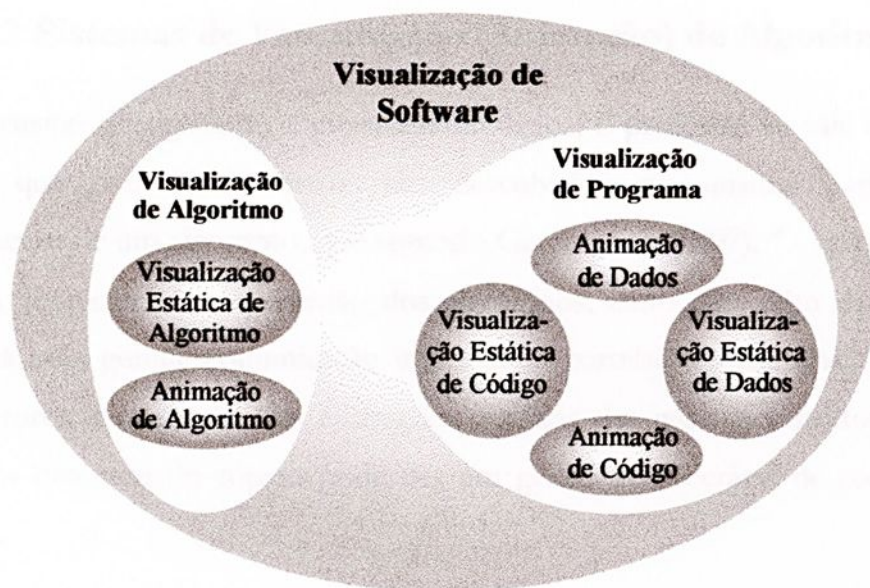


Figura 2.2 – Diagrama relacionando os termos utilizados em Visualização de Software
(PRICE et al., 1993, 1998)

Já *visualização de algoritmo*, segundo os autores supracitados, é a visualização de uma descrição de alto nível de uma parte de um programa que contrasta com a visualização de código ou dado (ambos dentro do domínio de visualização de programa).

O que diferencia a visualização estática da animação é a forma como este processo é apresentado. Por exemplo, a visualização estática de código (visualização de programa) pode ser estabelecida através de uma simples análise do código fonte escrito pelo programador, enquanto a animação deste mesmo código permite acompanhar a execução do programa destacando a linha que está sendo executada num determinado momento. De forma similar, a animação estática de algoritmo pode se dar através de esquemas, por exemplo, de “caixas e setas” representando uma lista encadeada, mostrando o seu conteúdo, que são esquematizadas num papel e redesenhadas para demonstrar uma transição. Na animação do algoritmo, estas mesmas “caixas e setas” são animadas de forma que as setas e conteúdo das caixas mudam dinamicamente, à medida que o programa vai sendo executado.

Nas seções subsequentes, sistemas de visualização de algoritmo e de programas são abordados com maiores detalhes, seguidos pela apresentação de alguns exemplos.

2.2.2 Sistemas de Visualização (Animação) de Algoritmos

No ensino de algoritmo e estruturas de dados, o professor se vale de esquemas e ilustrações que convencionalmente são desenhados no quadro para explicar o comportamento de um algoritmo, que segundo Garcia et al. (1997), “... contribuem muito pouco para aumentar a compreensão dos algoritmos, dado que, além das informações *depictadas*, há uma grande comunicação implícita na correlação entre elas...”. Segundo os mesmos autores, a animação das ilustrações estáticas das quais o instrutor se vale para explanar um determinado tópico pode dar um ganho considerável de comunicação da informação.

Através da visualização de um algoritmo, mais especificamente da sua animação, é possível visualizar o seu comportamento durante a execução. Para este fim, é criada uma abstração dos dados e das operações apresentadas por este algoritmo. Os dados (estado atual do algoritmo – valores atuais das variáveis) são mapeados para formar uma imagem que sofre alterações, uma vez que seus conteúdos são atualizados (transição de um estado para outro), gerando a animação do algoritmo.

Um sistema de animação de algoritmos é composto por um conjunto de ferramentas multimídia integradas (gráficos, animações, texto e vídeo) que simulam o funcionamento de um algoritmo abstraindo-se dos dados, das operações e da semântica do programa que existe por trás do algoritmo em questão.

Dessa forma, é possível oferecer facilidades para o usuário (aprendiz de programador) visualizar e interagir com a execução de um algoritmo, e para programadores (professores) desenvolverem as animações desses algoritmos em questão. O objetivo principal desta categoria de ferramentas é incutir a idéia do funcionamento de um determinado algoritmo, embora a compreensão do código que o implementa também tenha a sua importância.

Nesta categoria encontram-se ferramentas que permitem ao professor criar animações para explicar o funcionamento de um determinado algoritmo (estruturas de dados complexas, ordenação e classificação). Algumas mais sofisticadas permitem a animação dos algoritmos implementados pelos próprios aprendizes.



2.2.3 Sistemas de Visualização de Programas

O processo de visualização de um programa consiste em produzir visões animadas da dinâmica apresentada pela execução do programa em questão.

Sistemas de visualização de programas ajudam programadores a compreenderem o comportamento implícito nos programas que estão desenvolvendo, com o intuito de que o produto desenvolvido esteja livre de erros e funcione da forma como foi projetado. Estes sistemas provêem o usuário com informações visuais de forma a propiciar-lhe uma compreensão maior da solução implementada no programa, permitindo que ele possa elaborar um modelo mental de sua execução.

Diferentes aspectos da execução de um programa podem ser capturados por um sistema de visualização. Estes incluem a visualização dinâmica do código ilustrando a execução de cada comando (seqüências, desvios, chamadas de sub-rotinas) e a visualização das estruturas de dados ilustrando o comportamento do programa (conteúdo das constantes e variáveis) até a visualização de detalhes do hardware (alocação e acesso à memória, uso dos registradores). Este último aspecto da visualização não apresenta uma relevância tão grande quando o objetivo é depurar e corrigir um programa.

Segundo Price et al. (1993) muitos dos ambientes de programação visual oferecem componentes para visualização de programas, no entanto, a disponibilização destes recursos não é tão explícita e a sua utilização não se dá de uma forma tão direta como nos sistemas especialmente projetados para esse fim. Alguns desses sistemas podem incorporar recursos tanto de visualização de programas como de animação de algoritmos.

2.3 Exemplos de Sistemas de Visualização de Software

Um dos primeiros sistemas de visualização de softwares, considerado como um dos precursores desta área de pesquisa, foi produzido na década de oitenta pela Universidade de Toronto - **SOS** (abreviação para Sorting Out Sorting) (BAECKER, 1998; PRICE et al., 1993). Trata-se de um filme de 30 minutos animado através de computação gráfica que explica como nove diferentes algoritmos de classificação manipulam seus dados.



A partir de então os sistemas foram evoluindo e atualmente conta-se com uma gama bastante diversificada. Nas seções subseqüentes são apresentados exemplos, alguns dos quais foram incluídos no trabalho devido à sua importância histórica e outros por serem amplamente divulgados na literatura.

2.3.1 Balsa

O primeiro sistema de visualização de software interativo de expressão significativa foi o **BALSA** - Brown University Algorithm Simulator and Animator (BROWN, 1989; PRICE et al., 1993). Trata-se de um sistema de animação de algoritmo implementado em linguagem C, animando algoritmos codificados em Pascal. Sua versão inicial era executada sobre estações de trabalho (*workstation*) Apollo em monitores monocromáticos; a versão subseqüente passou a ser executada em microcomputadores Macintosh da Apple, recebendo o nome de **MacBALSA** (BROWN, 1989).

O domínio coberto pelo BALSA abrange algoritmos de classificação, grafos e estruturas de dados como listas, pilhas, filas e árvores.

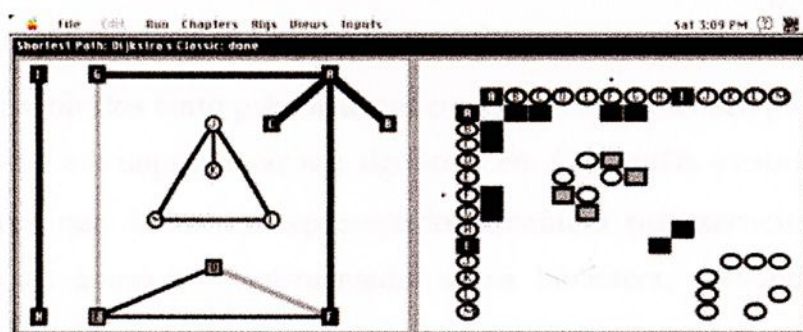


Figura 2.3 – Tela com a animação do algoritmo do caminho mais curto (Dijkstra)⁴

2.3.2 Zeus

O sistema **Zeus** (BROWN, 1991) foi uma evolução do BALSA. Este sistema de animação de algoritmos foi escrito em linguagem de programação Módulo-3 para estações de trabalho DEC. Basicamente, ele incorporou todos os recursos do BALSA acrescidos de

⁴ Fonte: <http://www.birkhauser.com/hypermedia/cyb157.gif>

recursos para animar programas paralelos e mecanismos de áudio (usando sintetizadores MIDI).

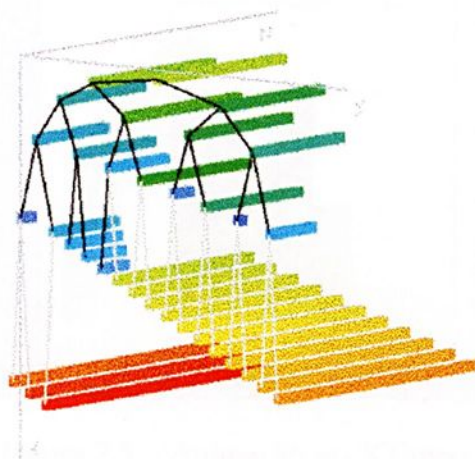


Figura 2.4 – Animação do algoritmo HeapSort em três dimensões⁵

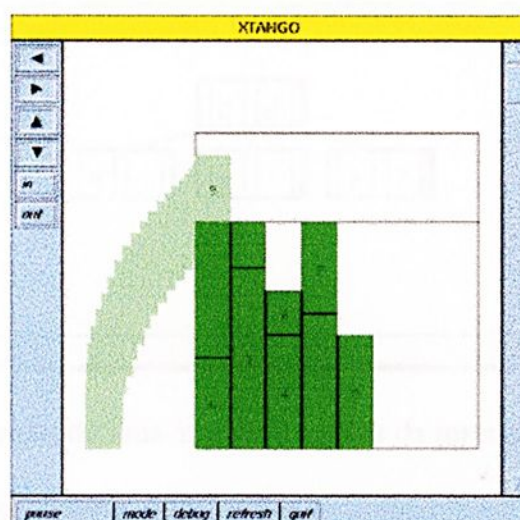
2.3.3 XTango

O XTango, sucessor do Tango (STASKO, 1993; PRICE et al., 1993) que foi um sistema bastante difundido na literatura, foi desenvolvido no College of Computing Georgia Tech, por John Stasko⁶ e colaboradores. É um sistema que suporta a animação de algoritmos desenvolvidos tanto pelo instrutor como pelo aprendiz. Seu processo básico de animação consiste em implementar um algoritmo em C e, então, estabelecer os eventos mais importantes que devem ser representados durante a sua execução. Estes eventos ativam rotinas de animação implementadas numa biblioteca, utilizando o pacote de animação do XTango para criar e manipular objetos (círculos, quadrados, linhas etc.).

O XTango opera em ambiente Unix e X11 Window System. Esta ferramenta pode ser adquirida via ftp anônimo no endereço <ftp.cc.gatech.edu> na pasta [pub/people/stasko](ftp://ftp.cc.gatech.edu/pub/people/stasko). Ele é distribuído com um grande pacote de animações (árvores AVL, *bubble sort*, *heap sort*, transformada rápida de Fourier, entre outros).

⁵ Fonte: <http://www.research.digital.com/SRC/zeus/heapsort3D.4.gif>

⁶ Página pessoal: <http://www.cc.gatech.edu/~john.stasko/>

Figura 2.5 – Animação no XTango ⁷

2.3.4 Astral

O ambiente Astral é uma ferramenta de animação de algoritmos desenvolvida no Instituto de Computação da Unicamp que, segundo seus autores (GARCIA et al., 1997), segue a abordagem *construtiva* no ensino de estrutura de dados – o uso de mecanismos de animação para auxiliar a implementação de algoritmos pelos próprios aprendizes (veja as figuras 2.6 e 2.7).

O ambiente foi desenvolvido para a plataforma Macintosh (sistema operacional Mac/OS v.7.5) devido à facilidade oferecida pela sua arquitetura para manipular objetos gráficos. A linguagem escolhida para implementar o código a ser animado foi Pascal, justamente por se adequar muito bem para fins didáticos (simplicidade da sintaxe, fortemente *tipada*, altamente estruturada e modular).

Através do Astral, além de o estudante ter acesso a aplicativos exemplos providos de animações gráficas que ilustram o funcionamento de algoritmos, ele também realiza suas próprias animações durante o processo de implementação da estrutura e dos algoritmos de maneira simplificada, mas com visualização gráfica notável e que, de acordo com experiências realizadas pelos autores, têm apresentado resultados sensivelmente positivos na aprendizagem de estrutura de dados.

⁷ Fonte: <http://www.cc.gatech.edu/gvu/softviz/algoanim/bpack.gif>

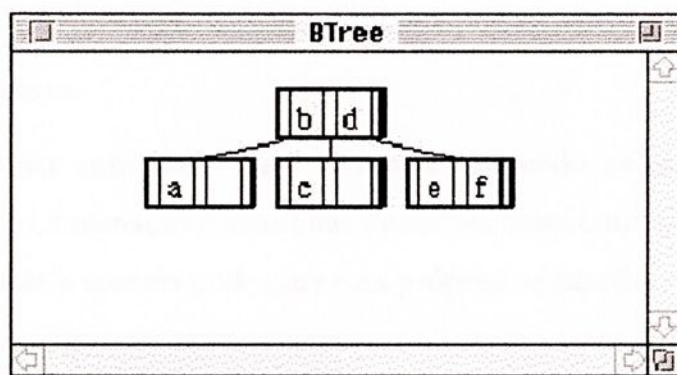


Figura 2.6 – Animação de uma árvore B - antes da inserção do elemento “g”⁸

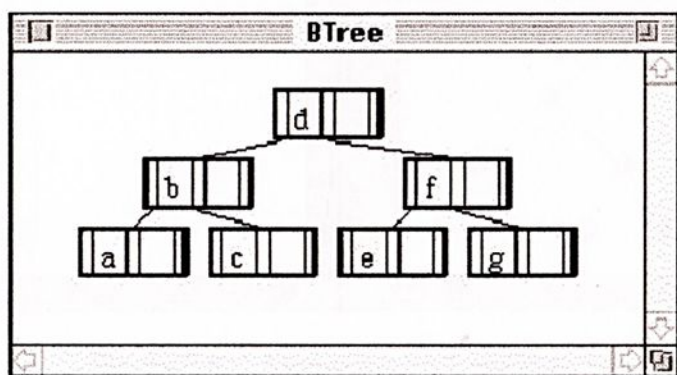


Figura 2.7 – Animação de uma árvore B - após a inserção do elemento “g”⁹

2.3.5 Leonardo

Desenvolvido no Departamento de Informática da Universidade de Roma, Leonardo (CRESCENZI, 2000) é um ambiente integrado para desenvolvimento e visualização de programas desenvolvido em linguagem de C. Sua concepção destina-se à aprendizagem da linguagem em questão, permitindo ao usuário depurar (*debugar*) visualmente seus programas e criar animações de algoritmos. A plataforma de execução baseia-se num Macintosh com sistema operacional Mac/OS v.7.5.

Através do Leonardo é possível editar, compilar e executar programas arbitrários na linguagem C (conta-se com todas as funções da biblioteca ANSI C), sem restrições. O

⁸ Fonte: <http://www.inf.ufsc.br/sbc-ie/revista/nr1/pict17.jpg>

⁹ Fonte: <http://www.inf.ufsc.br/sbc-ie/revista/nr1/pict18.jpg>

Leonardo oferece um ambiente integrado, onde a execução do programa pode ser visualizada passo a passo.

Além de contar com uma vasta biblioteca contendo animações de algoritmos clássicos (classificação, ordenação e estruturas de dados, entre outros), também conta com um editor gráfico onde o usuário pode criar suas próprias animações.

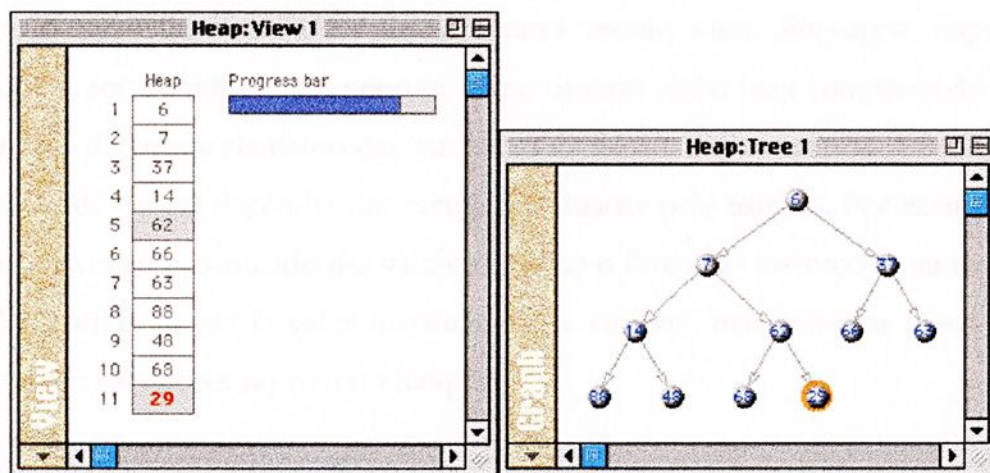


Figura 2.8 – Animação no Leonardo ¹⁰

2.3.6 Alma

Desenvolvido por pesquisadores do Departamento de Informática da Universidade do Minho (Portugal), o Alma é um sistema de visualização de programa implementado na linguagem Java para operar inicialmente sobre o sistema operacional Windows95. Apresenta como principais objetivos:

- ensino da programação através da análise de programas / algoritmos,
- ensino da matemática através da visualização detalhada dos cálculos de um programa,
- análise de respostas a problemas, visualizando a solução gerada pelos programadores a serem avaliados.

¹⁰ Fonte: <http://www.dis.uniroma1.it/~demetres/Leonardo/ImageGallery/images/heap.gif>

A finalidade do sistema proposto é analisar um texto de entrada (que poderá ser um programa clássico, um algoritmo, um documento etc.) e extrair a informação considerada importante que, depois de visualizada de uma forma clara e sugestiva, permite entender melhor o significado do texto fonte.

Se o texto de entrada for um programa escrito numa linguagem imperativa, a informação a ser exibida deverá permitir que o usuário tenha uma compreensão do fluxo de controle e do comportamento das estruturas de dados. A forma como essa informação será visualizada deverá depender das escolhas efetuadas pelo usuário. Por exemplo, pode-se optar por ver só o conteúdo das variáveis, ou só o fluxo das instruções, ou ambos (veja figura 2.9); poder-se ver o valor instantâneo da variável, mas também a sua evolução representando os valores no eixo do tempo.

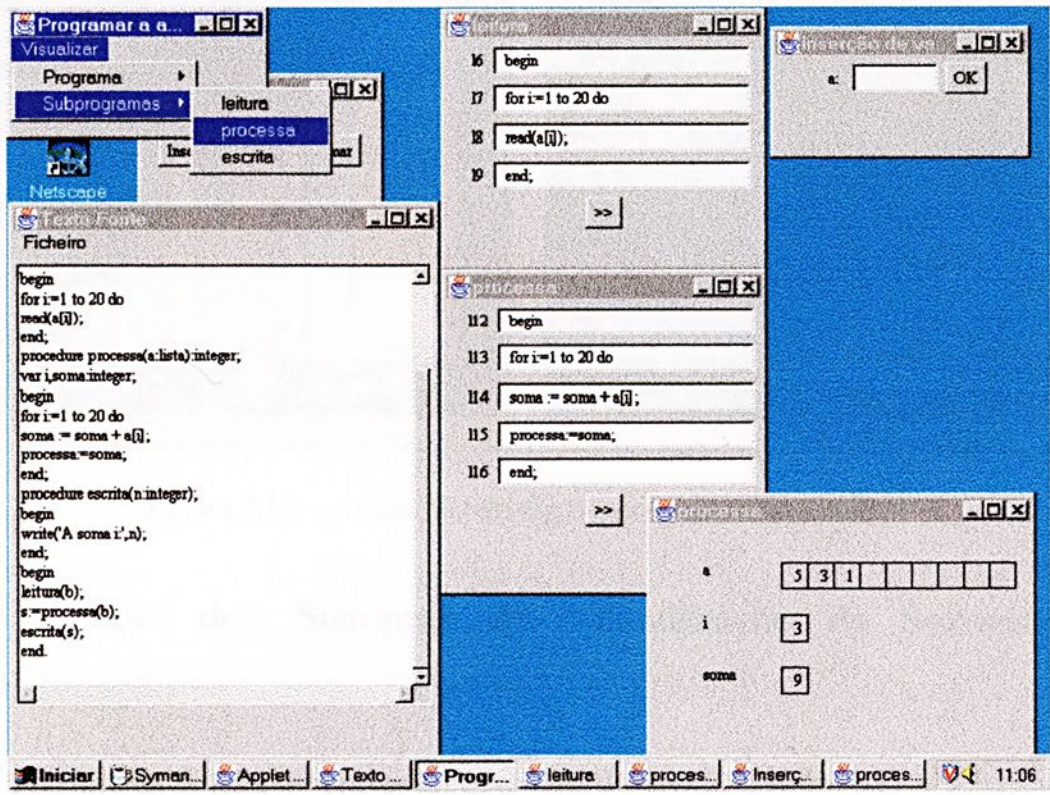


Figura 2.9 – Animação de um programa em Pascal ¹¹

¹¹ Fonte: <http://automatix.inesc.pt/CGME99/actas/c2/sec3.html>



2.3.7 Polka

O Polka é um outro sistema de animação de algoritmos desenvolvido no Georgia Institute of Technology (Graphics, Visualization & Usability Center) para animação de programas e algoritmos (em particular, programas paralelos e distribuídos). Derivado do XTango, é mais sofisticado e flexível: oferece recursos com alto nível de abstração para criação de animação de uma forma mais simplificada. Implementado em C++ para operar sobre o sistema operacional Unix (X11 Windows System), o Polka conta com uma diversidade de bibliotecas com animações (veja figura 2.10).

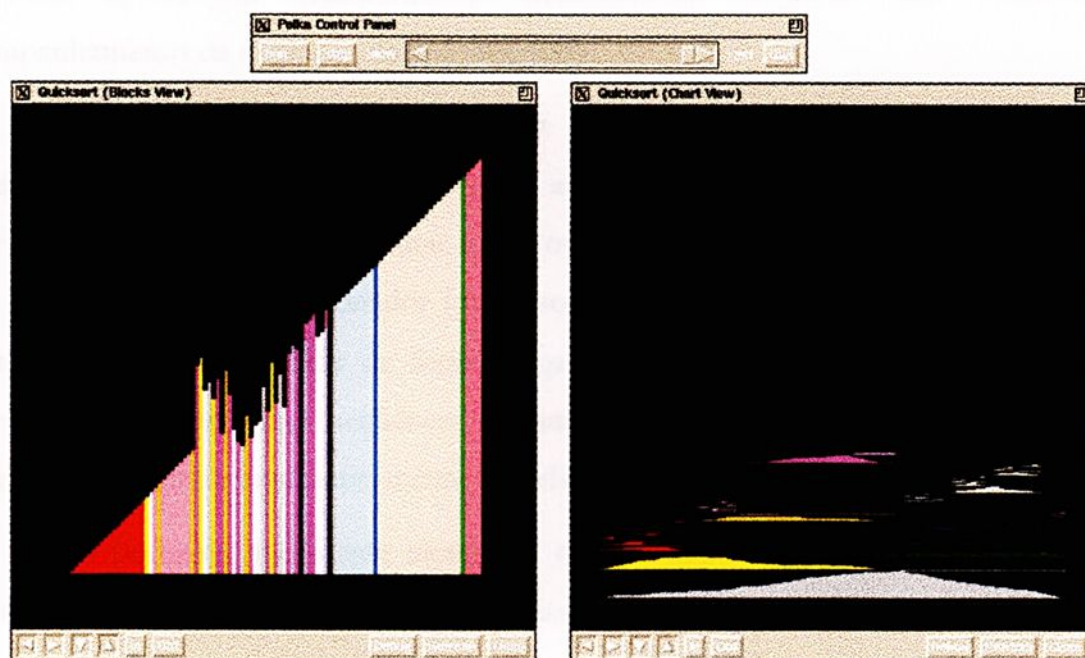


Figura 2.10 – Animação do algoritmo *quicksort* paralelo ¹²

2.4 O Papel dos Sistemas de Visualização de Software na Educação

Da mesma forma que programadores profissionais, professores e estudantes de ciência da computação freqüentemente recorrem ao uso de esquemas e figuras como auxílio na concepção, especificação e implementação de softwares (BAECKER, 1998, p. 369).

¹² Fonte: <http://automatix.inesc.pt/CGME99/actas/c2/sec3.html>

Professores inundam o quadro negro com esquemas e diagramas representando estrutura de dados e fluxo de controle e, despendem seu esforço e tempo apagando e redesenhando parte do esquema para simular o programa em funcionamento.

Semelhantemente, estudantes se esforçam para aperfeiçoar a compreensão do comportamento de um programa esquematizando representações de chamadas de procedimento, escopo de variáveis, alocação de memória, ligações entre ponteiros e organizações de registros.

Particularmente, estes esquemas e diagramas aos quais professores e estudantes recorrem apresentam características inerentemente estáticas que dificultam o acompanhamento da execução de um programa.

A exibição do comportamento de um programa requer uma representação altamente dinâmica, dificilmente conseguida através dos meios convencionais: esquemas e desenhos são imprecisos; representar todos os passos da execução requer muito tempo, o que acaba forçando o programador (professor e/ou estudante) a saltar e ignorar algumas passagens que podem influir de forma negativa no resultado final da execução. Sendo assim, seria útil ter uma seqüência de animações retratando o comportamento do programa construído automaticamente à medida que sua execução vai avançando.

Isto pode ser facilmente alcançado com o uso de sistemas de visualização de software; no entanto, o processo de visualizar um software para fins pedagógicos não é trivial, pois há numerosos detalhes complexos envolvidos num programa de computador que devem ser considerados. Para ser efetiva, a visualização do programa deve abstrair ou destacar aspectos essenciais de um programa: decidir qual código do programa e quais dados representar, como eles devem ser visualizados e quando atualizar suas representações durante a execução do programa. Estes são pontos fundamentais que devem ser previstos numa ferramenta de visualização de software. Mais importante ainda é tentar realçar os aspectos relevantes e suprimir os detalhes que não estejam intimamente relacionados com a lógica da programação, de forma a deixar claros, ordenados e atraentes os gráficos e as animações, e também determinar apropriadamente o ritmo e a velocidade da execução dos passos (BAECKER, 1998, p. 370).

2.4.1 As Diversas Formas de Utilização

Levando-se em consideração todos os detalhes supracitados, sistemas de visualização de software podem se apresentar como uma valiosa ferramenta, pois oferecem vantagens distintas comparadas à abordagem tradicional de ensino baseada em livros-textos, quadro negro e anotações em cadernos (GLOOR, 1998, p. 415). De acordo com Stasko e Lawrence (1998, p. 419), as formas mais usuais de aplicação destes recursos são:

- utilização em sala de aula como forma de auxílio ao professor para a explanação de conceitos e exemplos;
- utilização formal em aulas de laboratório, onde os alunos interagem com a ferramenta;
- utilização informal (fora do horário de aula) para os alunos aprofundarem-se mais no assunto abordado.

Segundo Bazik et al. (1998, p. 384), o papel mais evidente para sistemas de visualização de software na educação é para demonstração em aula. O aspecto visual (animação) oferecido por essas ferramentas pode ser utilizado pelo professor na explanação de comportamentos dinâmicos de um algoritmo: ele pode executar a animação de um algoritmo de forma interativa para demonstrar, por exemplo, a chamada recursiva de uma sub-rotina.

Disponibilizar a animação a todos na sala de aula é um fator preponderante para a sua exposição. Isto pode ser realizado através de um projetor conectado ao computador ou mesmo num laboratório onde a animação visualizada no computador do professor seja transmitida de forma sincronizada a todos os outros computadores.

Um detalhe importante a ser destacado quando ferramentas se prestam ao auxílio na exposição das aulas está relacionado às questões cognitivas dos estudantes. Deve haver uma curta distância cognitiva entre o conceito e a visualização (BAZIK et al., 1998, p. 386). Quanto mais intimamente as imagens corresponderem ao modelo mental, maior será a efetividade da visualização na sala de aula. Quando o mapeamento é óbvio e direto, a carga cognitiva para os estudantes será menor e, conseqüentemente, a representação

torna-se transparente. Dessa forma, eles podem voltar sua concentração nas idéias sendo ilustradas, em detrimento da ilustração em si.

É desejável que a explanação de conceitos feita pelo professor em sala de aula seja acompanhada pela prática em laboratório onde os estudantes possam realizar atividades para aperfeiçoar a compreensão. O uso interativo de sistemas de visualização pelos estudantes permite a análise de exemplos e a realização de exercícios práticos de uma forma direta: usando animações de algoritmos interativamente, o estudante pode explorar as particularidades da sua execução, manipulando os dados de entrada e acompanhando as ações realizadas para gerar os resultados.

A utilização formal em aulas no laboratório pode ser realizada através de listas de exercícios com a supervisão do professor. Para permitir fácil interação, esses sistemas devem apresentar interfaces amigáveis, colocando de forma clara e consistente as ações que podem ser realizadas pelo usuário. Mesmo assim, pode haver certa carga cognitiva que dificulte ou cause dúvida aos estudantes, relacionada aos detalhes técnicos de utilização do sistema, o que torna importante a presença do professor para supervisionar a atividade.

A realização de atividades extra-classe deve ser estimulada pelo professor através de exercícios de fixação. O sistema de visualização pode ser utilizado informalmente pelo estudante fora do horário de aula. Para suportar esse tipo de utilização, o sistema deve contar com recursos que minimizem a necessidade da presença do professor. Este deve prover os estudantes com uma ampla biblioteca ou tutorial, onde se podem acessar facilmente informações sobre a utilização do sistema e sobre os conceitos abordados pelo professor em sala de aula.

2.4.2 Benefícios Esperados no Processo de Ensino-Aprendizagem de Lógica de Programação

Com o desenvolvimento das pesquisas nesta área, um número grande de ferramentas tem sido concebido. Na teoria, apregoa-se que os benefícios a serem obtidos com a utilização dessas ferramentas serão bastante significativos; ao menos esta é a crença que todo desenvolvedor tem sobre sua ferramenta. No entanto, é necessário realmente verificar essa efetividade na prática através da realização de estudos empíricos.



Vários experimentos relatados por Gloor (1998) e, Stasko e Lawrence (1998) têm apontado uma série de benefícios obtidos com a utilização destas ferramentas; no entanto, na maioria dos casos houve uma superestimação das expectativas que não foram comprovados na prática.

Cada um dos experimentos, num total de três, apresentados por Stasko e Lawrence (1998) foi realizado em um pequeno grupo de estudantes dividido em duas turmas: numa foi aplicado o ensino convencional baseado em quadro negro e livro-texto; na outra foi utilizado um sistema de visualização de software tanto na exposição da aula quanto na prática de exercícios. Após a apresentação de um único tópico, os estudantes foram estimulados a realizarem exercícios e, logo após, foram submetidos a um teste de avaliação.

Através da análise destes experimentos os autores depreendem quatro importantes observações:

- O uso de sistemas de visualização de software apresenta seu valor pedagógico, mesmo com um grau mínimo de benefício, devido ao simples fato de prender a atenção dos estudantes que se sentem motivados e entusiasmados com a nova abordagem dada ao ensino de algoritmos.
- É um recurso que não está acessível a todos. O cenário que envolveu os estudos realizados conta com laboratórios onde cada aluno dispõe de um computador e um dispositivo de projeção que exhibe a tela do computador utilizado pelo professor. Seria difícil criar uma infra-estrutura que acomodasse todos os alunos de uma turma durante todo o período letivo.
- A habilidade do professor em conduzir a aula é fator determinante na obtenção de resultados mais significativos. Em algumas situações, uma parcela dos alunos que tentou inicialmente aprender um algoritmo sem se preocupar com o aspecto visual, posteriormente encontrou dificuldades em traduzir as animações para o algoritmo que estas estavam representando.
- De uma forma geral, o uso interativo do sistema pelos alunos em sessões de laboratório trouxe contribuições inestimáveis: a facilidade encontrada em fornecer

novos dados de entrada durante a execução do algoritmo estimulou os estudantes a testarem novas situações e a realizarem previsões sobre as ações subseqüentes; também estimulou os estudantes a criarem e modificarem os algoritmos estudados para adequá-los a novas situações não previstas inicialmente pelo professor.

A experiência relatada Gloor (1998) envolveu um cenário mais amplo e foi aplicada durante todo o período letivo na disciplina “Introdução a Algoritmos” numa turma do curso de Ciência da Computação. Gloor estabelece como principais benefícios obtidos pelos estudantes com o uso de sistemas de visualização de software os seguintes itens:

- motivação: a partir do momento em que os estudantes são confrontados com assuntos complexos através da apresentação de uma forma atrativa, estudantes sentem-se muito mais motivados a estudá-los;
- forma alternativa de experimentar os algoritmos: além da forma tradicional de experimentar os algoritmos através do tradicional “teste de mesa”, eles podem visualizar e estudar seus aspectos observando e interagindo com as animações;
- oportunidade para desenvolvimento de habilidades analíticas: esses sistemas auxiliam no desenvolvimento das habilidades analíticas permitindo que os estudantes possam interagir com os algoritmos, fornecendo novos dados para analisá-los e posteriormente reprojeta-los de forma otimizada, tomando-se por base a experiência anterior.

Apesar da efetividade relatada com base na realização de diversos estudos, todos os autores colocam de forma bem clara que o uso de sistemas de visualização de software não é uma panacéia; ele não resolve todos os problemas relacionados ao desenvolvimento de software, mas sem dúvida é um recurso do qual o professor pode se valer para, pelo menos, deixar os estudantes mais motivados e receptivos à introdução de conceitos mais complexos.

No capítulo a seguir, são apresentados alguns sistemas utilizados no ensino de algoritmos, na sua maioria sistemas de visualização de software.



Capítulo 3

Avaliação de Algumas Ferramentas que Auxiliam o Processo de Ensino-Aprendizagem em Lógica de Programação

Particularmente, sistemas de visualização de software englobam uma ampla gama de aplicações que abordam diversos aspectos do processo de desenvolvimento de um software. De uma forma geral, o uso destes sistemas pode ocorrer sob duas perspectivas:

- são utilizados no ensino de algoritmos e programação para demonstrar aos alunos como determinados algoritmos funcionam;
- servem de apoio ao programador para que ele possa ter uma compreensão maior do código implementado.

Uma vez que a finalidade deste trabalho é discutir o processo de ensino-aprendizagem de lógica de programação, a atenção é voltada às ferramentas de cunho didático, considerando apenas os sistemas de visualização de software que se prestam a este fim.

Várias ferramentas têm sido concebidas com o intuito de permitir a visualização da dinâmica inerente à execução de um programa em uma determinada linguagem de programação ou pseudocódigo.

Neste capítulo são estabelecidos alguns requisitos a serem considerados na implementação e avaliação de sistemas de visualização de software. Algumas ferramentas são descritas com um certo grau de detalhe, destacando suas potencialidades e

deficiências, sendo também apresentado um quadro comparativo entre elas, confrontando-as com os requisitos estabelecidos.

3.1 Estabelecimento dos Requisitos a Serem Considerados na Avaliação de Ferramentas que Auxiliam o Processo de Ensino-Aprendizagem de Lógica de Programação

Com base na análise de algumas ferramentas (veja seção 3.2) e na revisão da literatura, nesta seção são estabelecidos alguns requisitos a serem considerados na implementação e avaliação de sistemas que se prestam ao ensino de lógica de programação. É importante ressaltar que, neste processo, o objetivo não é ensinar a sintaxe da linguagem; a prioridade está em fazer com que o aprendiz compreenda o comportamento apresentado por seus construtores de modo que ele possa valer-se desta estrutura para implementar suas soluções (AZEREDO, 2000).

O estudo destes requisitos baseia-se no trabalho de Nakamura e Massi (2001) e destacam-se aspectos como paradigma de programação e gramática adotada na linguagem como fatores a serem considerados inicialmente. Outros aspectos determinantes na efetividade da ferramenta estão relacionados à facilidade de edição, compilação, execução e visualização dos algoritmos implementados pelo aprendiz. Cada um destes aspectos é discutido com detalhe nas seções subseqüentes.

3.1.1 Considerações sobre o Paradigma de Programação

Do ponto de vista do paradigma de programação (imperativo, orientado a objeto, funcional, lógico e paralelo), cada um impõe diferentes abstrações e entidades a serem visualizadas. A razão disso é que cada um apresenta um estilo diferente de computação e, conseqüentemente, uma abordagem diferente para a solução do problema.

O paradigma de programação imperativo é baseado na abordagem *procedural* de solução de problemas, estabelecendo blocos de operações para atuar sobre um conjunto de dados. Sendo assim, as abstrações necessárias para visualizar programas baseados neste

paradigma são: variáveis/estrutura de dados, procedimentos/funções e construtores de controle (estrutura condicional, estrutura de repetição etc.).

Os requisitos de visualização para programas que adotam o paradigma paralelo são basicamente os mesmos da contrapartida imperativa (seqüencial) acrescidos de abstrações que permitem visualizar *threads* e processos, e de aspectos de comunicação e sincronização entre processos (STASKO; KRAEMER, 1993). Quando seções de código de um programa são executadas em paralelo, a ordem relativa de execução entre elas pode causar resultados indesejados caso não sejam devidamente tratadas. Deve ser permitido ao usuário visualizar o momento em que ocorre a comunicação entre os processos para que ele possa precaver-se de situações não previstas, realizando a devida sincronização (OUDSHOORN et al., 1996).

Na visualização de programas que seguem o paradigma funcional os principais componentes a serem tratados são funções e estruturas de dados, mais especificamente listas simples ou conjuntos. O Kaestle (ELLERSHAW; OUDSHOORN, 1994) é um sistema de visualização de programas escritos em LISP. Ele provê o usuário com representações gráficas de uma estrutura de lista; todos os objetos de dados criados pelo programa sendo visualizado podem ser exibidos com seus valores sendo atualizados diante da ocorrência de cada evento definido pelo usuário.

O paradigma orientado a objeto impõe a visualização de seus construtores compreendendo basicamente uma biblioteca de classes pré-definidas (uma vez que a reutilização de código é uma das principais preocupações do paradigma orientado a objeto) e classes criadas pelo usuário. A visualização também deve se estender à criação de objetos, ao método sendo invocado atualmente e ao objeto com o qual está interagindo, à herança e ao encapsulamento, entre outros (WIDJAJA; OUDSHOORN, 1997).

Para visualizar programas lógicos, são necessárias abstrações como cláusula e unificação. Quando visualizadas, estas abstrações devem produzir visões de cláusulas, conexões lógicas E/OU, unificações e chamadas recursivas (OUDSHOORN et al., 1996). Em Mulholland (1998) é apresentada uma ferramenta que permite visualizar programas escritos em Prolog.

Os demais requisitos analisados são inerentemente de sistemas de visualização que seguem o paradigma de programação imperativo. Isto se deve ao fato dessa abordagem ser a mais amplamente utilizada para iniciar o estudante na atividade de programação (TIMÓTEO; BRASILEIRO, 1998).

3.1.2 Considerações sobre a Gramática da Linguagem Algorítmica

Basicamente, um algoritmo (programa) é constituído por estruturas de dados e estruturas de controle (HAREL, 1992). As estruturas de dados representam os dados manipulados (entrada, saída e representações internas) pelo algoritmo; na sua essência englobam variáveis simples e variáveis compostas (vetores e registros). As estruturas de controle determinam o fluxo da execução do algoritmo e englobam basicamente seqüência, seleção, iteração e modularização.

É desejável que a gramática adotada na implementação da linguagem algorítmica, além de apresentar uma sintaxe simples, comporte os construtores básicos (estrutura de dados e estrutura de controle) de um algoritmo.

Uma gama bastante ampla de elementos adicionais pode ser incorporada a estes construtores (recursividade, ponteiros, operações com cadeia de caracteres, arquivos, bibliotecas – rotinas pré-definidas – entre outros).

3.1.3 Suporte a Edição e Compilação do Algoritmo

Seria interessante que, ao editar o algoritmo, o usuário tivesse acesso a um tutorial que o instrísse sobre a utilização de cada construtor da linguagem e que exibisse também a sua sintaxe. Tal fato o desobrigaria de recorrer a material didático auxiliar ou às anotações para esclarecer alguns pontos obscuros durante a implementação do algoritmo. A utilização de um tutorial baseado em hipertexto/hipermídia, além de propiciar uma consulta rápida ao estudante, pode encorajá-lo a explorar assuntos adicionais (Gloor, 1998).

Gabaritos para edição de comandos permitem ao aprendiz voltar sua concentração na busca da solução em detrimento dos detalhes de sintaxe da linguagem. O usuário poderia selecionar o construtor desejado através de um menu, e um gabarito seria exibido para que alguns argumentos fossem digitados; automaticamente o código referente à implementação do construtor em questão seria inserido no ponto desejado.

Um dos grandes problemas apresentados pelos compiladores de uma linguagem de programação, de uma forma geral, diz respeito às mensagens de erros: estas geralmente são vagas e/ou codificadas, dificultando a identificação exata do erro cometido pelo programador. Considerando-se uma ferramenta para fins didáticos, uma atenção maior deve ser dada a este fato, de forma a prover o usuário de mensagens significativas e precisas.

3.1.4 Visualização de Código e de Dados

Um fator indispensável na compreensão do algoritmo (e na correção de possíveis erros) é permitir que o aprendiz (programador) possa acompanhar a execução de um algoritmo visualizando o código em execução e o conteúdo dos dados sendo manipulados.

A visualização do código é um dos fatores mais importantes a serem considerados na compreensão do “programa” (ROSENBERG, 1996, p.24). Essa visão dá ao programador a idéia exata da instrução sendo executada num determinado instante, destacando a linha do código fonte escrito pelo programador.

Através da visualização dos dados, deve ser possível ao programador avaliar o comportamento do programa, analisando o conteúdo apresentado por suas variáveis e constantes (uma lista com o seu identificador e seu respectivo conteúdo deve ser exibida). Tal situação torna-se indispensável quando se deseja verificar o funcionamento correto de uma implementação. Durante o acompanhamento da execução, cada atualização feita nas variáveis deve resultar na alteração do respectivo conteúdo que é exibido com o novo valor.

Nos *debuggers* que acompanham pacotes de ambientes para desenvolvimento de programas, a visualização dos dados é controlada pelo usuário, ou seja, é ele quem adiciona e remove as variáveis cujo conteúdo deseja visualizar. Ambientes de cunho

didático como os que estão sendo discutidos devem abstrair-se desses detalhes para propiciar ao usuário uma interação mais “amigável”. Isto pode ser feito adicionando-se automaticamente à lista todos os dados que fazem parte do escopo da execução do algoritmo num determinado momento (variáveis/constantes globais e variáveis/constantes locais se uma sub-rotina estiver sendo executada no momento) e removendo-os, uma vez que deixem de fazer parte do escopo da execução (variáveis/constantes locais de uma sub-rotina cuja execução se encerrou).

3.1.5 Controle da Velocidade de Execução

Uma vez que o algoritmo já passou pelo processo de compilação, o usuário pode executá-lo para verificar se sua implementação funciona corretamente. Com o intuito de facilitar a realização desta tarefa, seria interessante prover o usuário de mecanismos que lhe permitam controlar a velocidade da execução de um algoritmo de acordo com o seu ritmo de assimilação. A utilização de recursos como execução direta, execução passo a passo, breakpoints e execução reversa aliada à visualização de código e de dados oferecem ao aprendiz a compreensão da dinâmica apresentada pela execução do algoritmo desenvolvido e lhe permite depurar e corrigir possíveis erros (lógicos) que possam ter ocorrido na sua concepção.

A execução direta ocorre num único passo do início ao fim: as únicas interações com o usuário serão na entrada e na saída de dados, durante a execução dos respectivos comandos.

Para permitir uma compreensão maior, a opção pela execução passo a passo torna-se indispensável (este é um recurso comumente presente nos *debuggers* convencionais), uma vez que, com este recurso o usuário acompanha a execução do programa linha a linha, seguindo o seu fluxo de controle.

Os breakpoints são recursos críticos que permitem ao programador controlar a execução do programa e especificar onde a aplicação deve parar para permitir uma análise mais detalhada. Um *breakpoint* está vinculado a uma linha do código fonte e determina que a execução seja realizada até o ponto onde o mesmo foi estabelecido e, a partir daí, o usuário pode alternar para a execução passo a passo, de forma a analisar com maiores

detalhes um determinado trecho da sua implementação. É desejável que se possa inserir e remover quantos *breakpoints* forem necessários.

Um outro recurso de muita valia para uma ferramenta de cunho didático é a “execução reversa” (BORONI et al., 1996), geralmente ausente nos *debuggers* convencionais. Através deste recurso seria possível desfazer a execução (passo a passo) de um trecho e refazê-la - este aspecto torna-se interessante, uma vez que o aluno não precisará reiniciar a execução caso não tenha compreendido o funcionamento de uma determinada passagem.

3.2 As Ferramentas Avaliadas

Fazem parte desta avaliação cinco ferramentas utilizadas com o propósito final de ensino em lógica de programação. A escolha destas deve-se ao fato de serem facilmente obtidas via internet ou estarem disponíveis como ferramentas que acompanham livros direcionados ao ensino em questão.

Além das ferramentas de visualização de software, foi considerada uma outra categoria que engloba ferramentas que permitem a execução de programas escritos em pseudocódigo utilizando termos em português e, que apesar de despojado do aspecto visual, oferece uma contribuição valiosa no processo de ensino-aprendizagem de lógica de programação.

As cinco ferramentas adotam o paradigma imperativo de programação. De acordo com uma pesquisa realizada por Timóteo e Brasileiro (1998), esse paradigma se apresenta como uma unanimidade quase absoluta como o primeiro a ser abordado no ensino de programação.

As ferramentas consideradas nesta avaliação são:

- **Portugol/Plus** (ESMIN, 1998): ferramenta desenvolvida por Ahmed Ali Abdalla Esmín (página pessoal: <http://inf.ulbrajp.com.br/~ahmed/index.htm>); pode ser solicitada diretamente ao seu autor através do e-mail ahmed@ulbrajp.com.br;
- **ILA** (Interpretador de Linguagem Algorítmica) (CRESPO; EVARISTO, 2000): Desenvolvido por Sérgio Crespo (página pessoal <http://www.exatec.unisinos.br/~crespo/>), esta ferramenta é parte integrante do livro

“Aprendendo a Programar: Programando numa Linguagem Algorítmica Executável (ILA)” do mesmo autor e pode ser adquirida via *download* no endereço <http://www.inf.unisinos.br/~crespo/ila/ila.htm>;

- **DynaLab** (BIRCH et al., 1995; BORONI et al., 1996): ferramenta desenvolvida no departamento de Ciência da Computação da Universidade do Estado de Montana (EUA) por Rocky Ross e colaboradores (pode ser adquirida via *download* na página <http://esus.cs.montana.edu/~dynamlab/download/form.html>);
- **PascalX**: desenvolvida por Athur Vargas Lopes (página pessoal: <http://www.ulbra.tche.br/~avl/home.htm>), pode ser adquirida via *download* no endereço <http://www.ulbra.tche.br/~avl/pascal1.htm>.
- **Construtor**: desenvolvida pelo Centro Educacional de Informática Aplicada do SENAC (Rio de Janeiro), esta ferramenta acompanha o livro “Construção de Algoritmos” (FERNANDES; BOTINI, 1998).

3.2.1 Portugol/Plus

Tanto o Portugol/Plus como o ILA (veja seção 3.2.2), por não possuírem um ambiente para execução e visualização das implementações do estudante, não são classificados como sistemas de visualização de software. No entanto, por constituírem ferramentas que podem contribuir de forma significativa no ensino de lógica de programação, foram considerados neste trabalho.

Do ponto de vista funcional, ambos se prestam para o ensino de lógica de programação, permitindo que o usuário execute as implementações escritas num determinado pseudocódigo.

O Portugol/Plus é uma ferramenta desenvolvida para operar em microcomputadores padrão IBM-PC e compatíveis com o sistema operacional DOS (*Disk Operating System*) e oferece ao usuário um ambiente para edição e compilação do algoritmo, cujo pseudocódigo é baseado na linguagem Pascal. Esse ambiente é composto por um editor de texto convencional (padrão ASCII - veja figura 3.1) acrescido de funções para compilar e executar os algoritmos do estudante.



O compilador do Portugol/Plus é bastante peculiar (veja figura 3.2): ele rastreia o algoritmo escrito em pseudocódigo (arquivo texto com extensão “.POR”) para verificar a existência de erros léxicos e sintáticos e, gera um programa equivalente na linguagem Pascal (com extensão “.PAS”) e se vale do compilador do Turbo Pascal (TPC) para que este último seja compilado e executado.

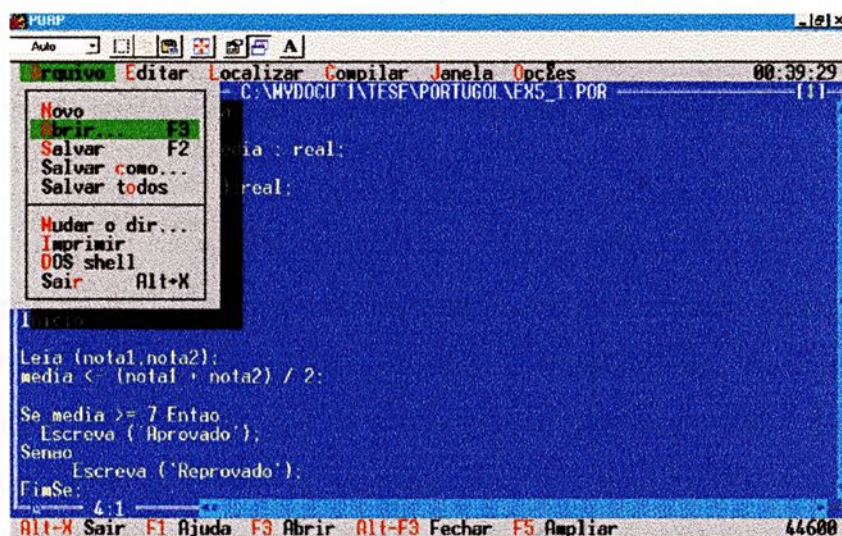


Figura 3.1 – Ambiente de desenvolvimento do Portugol/Plus

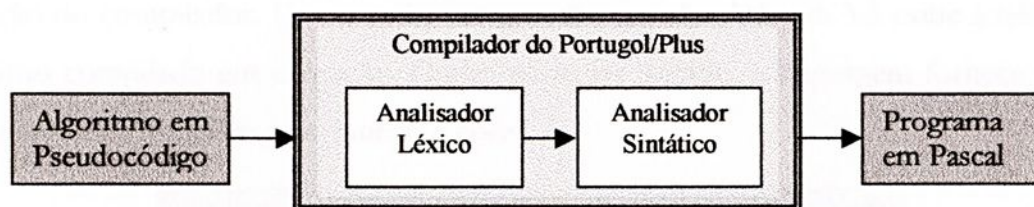


Figura 3.2 – Processo de compilação do Portugol/Plus (Esmin, outubro de 1998)

Uma grande vantagem do uso desta ferramenta está no fato de colocar o aprendiz em contato com o rígido formalismo apresentado pelas linguagens de programação. Por se tratar de uma ferramenta relativamente simples, não oferece recursos para visualizar (acompanhar passo a passo) a execução do algoritmo.

Um aspecto deficiente apresentado pelo Portugol/Plus diz respeito à sua documentação: ele não oferece tipo algum de ajuda para esclarecer a sintaxe dos comandos suportados por sua linguagem.

A seguir, são apresentadas algumas características e restrições da linguagem algorítmica suportada pelo Portugol/Plus:

- sua gramática é uma tradução direta da linguagem Pascal;
- não suporta o uso de bibliotecas e funções externas;
- suporta vetor e registro; não suporta a implementação de arquivos;
- permite a implementação de sub-rotinas (procedimentos e funções) com passagens de parâmetro por valor e referência, suportando inclusive a recursividade;
- não suporta manipulação de ponteiros e alocação de variáveis dinâmicas.

3.2.2 ILA

A ferramenta ILA (Interpretador de Linguagem Algorítmica) apresenta as mesmas funcionalidades do Portugol/Plus. No entanto, ela adota uma linguagem algorítmica bastante particular e simples. O usuário aprendiz deve utilizar um editor de texto qualquer para editar seus algoritmos. Uma *ajuda* na forma de tutorial explica a sintaxe de toda a linguagem. O algoritmo digitado deve ser compilado via *prompt* do DOS passando-se o nome do arquivo editado (com extensão “.ILA”) como argumento na chamada de execução do compilador. Um arquivo executável é gerado. A figura 3.3 exibe a tela de um algoritmo compilado em execução. Como pode ser notado, a linguagem fornece suporte para o aprendiz trabalhar com janelas e cores.

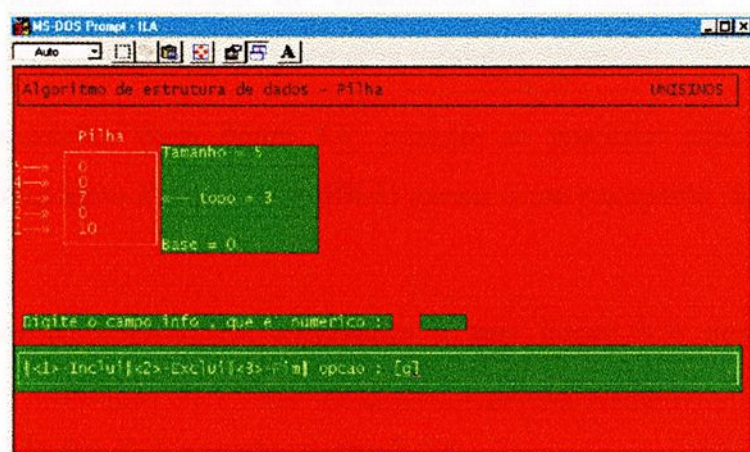


Figura 3.3 – Execução de um programa desenvolvido com o ILA

Algumas características e restrições da linguagem suportada pelo ILA incluem:

- não suporta o uso de bibliotecas e funções externas;

- suporta vetor, mas não permite a implementação de registros nem de arquivos;
- permite a implementação de sub-rotinas (somente função) com passagem de parâmetro somente por valor, suportando chamada recursiva;
- não permite a manipulação de ponteiros nem alocação de variáveis dinâmicas.

3.2.3 DynaLab

Um grupo de pesquisa do Departamento de Ciência da Computação da Universidade do Estado de Montana tem trabalhado num grande projeto denominado **DYNALAB** (uma abreviação para *DYNAmic LABoratory*). Trata-se de um sistema que suporta demonstrações interativas altamente visuais de experimentos em ciência da computação (BIRCH, 1995; BORONI et al., 1996). A sua versão 1.0 consiste em um computador virtual denominado E-Machine – abreviação para Education Machine – (PATTON, 1989), um emulador para o E-Machine (BIRCH, 1990) e um compilador para a linguagem PASCAL (GOOSEY, 1993).

Trata-se de uma ferramenta para visualização de software que opera sobre o sistema operacional Windows95/NT, animando programas escritos na linguagem supracitada. Dentre as categorias da classificação proposta neste trabalho, esta ferramenta é classificada como um sistema de visualização de programas (veja seção 2.2.3). Basicamente, o DynaLab é composto por dois módulos distintos:

- O editor/compilador (veja figura 3.4) permite ao aprendiz editar seus programas utilizando a linguagem Pascal e compilá-lo gerando um programa objeto que pode ser animado.
- O visualizador (o coração da ferramenta – veja figura 3.5) permite ao aprendiz acompanhar a execução do programa (passo a passo) visualizando o código linha a linha e conteúdo dos dados sendo manipulados.

O fato de o DynaLab ser composto por dois módulos independentes pode acarretar um inconveniente, por dificultar a interação com o usuário.

Para utilizar o DybaLab, o usuário deve digitar seu programa (em Pascal) no módulo Editor/Compilador. Trata-se de um editor de texto bem simples, com opção para compilar o programa. Uma vez que o programa esteja livre de erros (léxico, sintático e semântico), o compilador gera um código objeto com extensão “.COD” (o código fonte apresenta extensão .PAS), e é a partir deste que as visualizações são geradas.

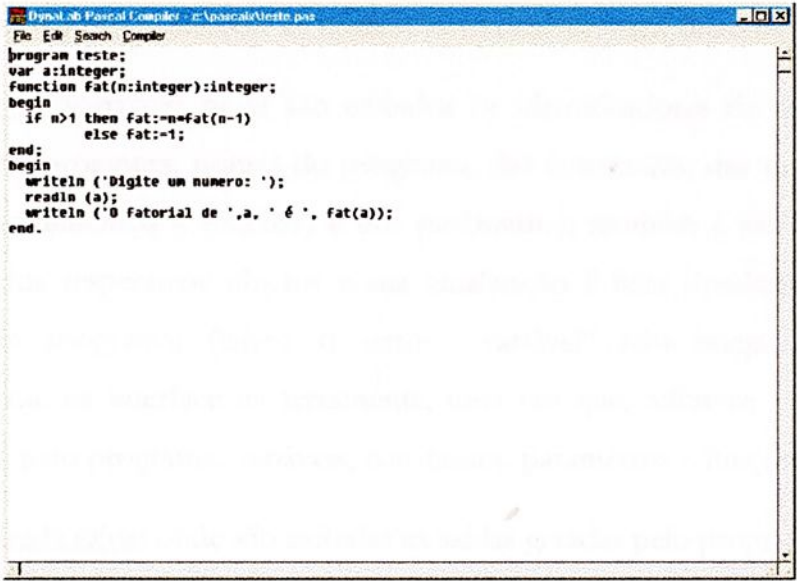


Figura 3.4 – Módulo editor/compilador do DynaLab

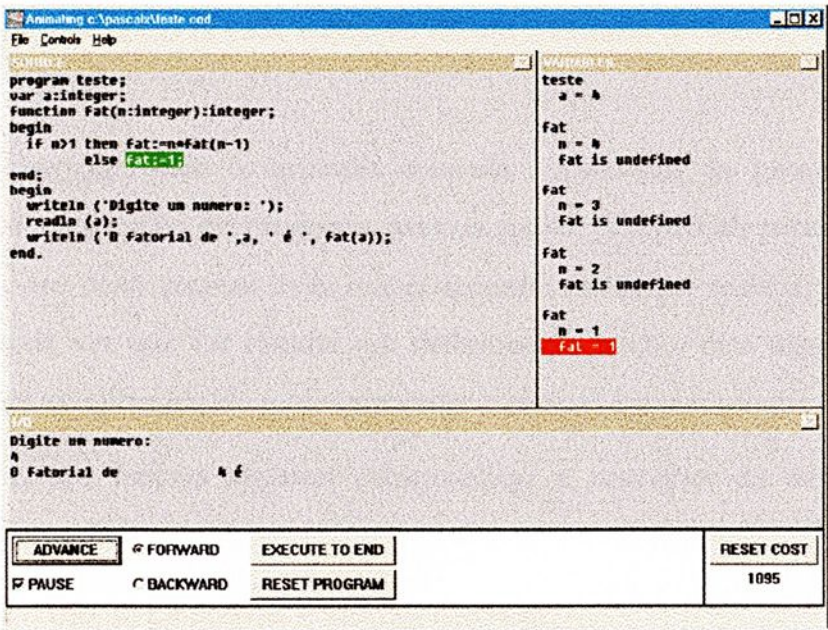


Figura 3.5 – Ambiente de execução e visualização do DynaLab

O visualizador (figura 3.5), por ser um módulo independente do editor/compilador, deve ser inicializado e o usuário deve abrir o arquivo objeto (.COD) para que o mesmo possa ser animado. Este módulo é composto por quatro janelas:

- visualizador de código: onde o código fonte do programa editado pelo usuário é exibido; na execução passo a passo do programa, a linha sendo executada é destacada em vídeo reverso possibilitando ao usuário acompanhar o seu fluxo de controle;
- visualizador de variáveis: onde são exibidos os identificadores de todos os objetos declarados no programa: nomes do programa, das constantes, das variáveis, das sub-rotinas (procedimentos e funções) e dos parâmetros; também é exibido o conteúdo dos dados dos respectivos objetos e sua atualização é feita simultaneamente com a execução do programa; (talvez o termo “variável” não esteja sendo utilizado adequadamente na interface na ferramenta, uma vez que, refere-se a todos os dados manipulados pelo programa: variáveis, constantes, parâmetros e funções);
- janela de entrada saída: onde são exibidas as saídas geradas pelo programa e as entradas fornecidas pelo usuário; pode-se optar por realizar a entrada de dados através do teclado (neste caso uma janela de leitura é exibida para que o usuário forneça os dados) ou de um arquivo texto (a cada execução de um comando de leitura, uma linha é lida do arquivo texto);
- painel de controle: onde o aprendiz controla a execução do programa; além de implementar a execução do programa passo a passo, o DynaLab permite a execução reversa – o uso deste recurso evita que o aprendiz tenha que reiniciar a execução do programa toda vez que for estudar um determinado trecho: caso alguns passos não fiquem claros ao aprendiz, ele pode retroceder e executá-los novamente.

A seguir, são listadas algumas características e restrições da linguagem Pascal suportada pelo DynaLab:

- não suporta o uso de bibliotecas e funções externas;
- suporta vetor e registro; não suporta a implementação de arquivos;



- permite a implementação de sub-rotinas (*procedures* e *functions*) com passagens de parâmetro por valor e referência, suportando chamada recursiva;
- permite manipulação de ponteiros com alocação de variáveis dinâmicas.

3.2.4 PascalX

PascalX é uma ferramenta que segue a mesma linha do DynaLab (sistema de visualização de programas). Trata-se de um compilador/interpretador para a linguagem Pascal que também permite o uso de vocabulário básico da linguagem em português (as palavras reservadas da linguagem Pascal são traduzidas para o português). O PascalX foi desenvolvido em C++ (Ansi e MS Visual C++) para operar sobre o ambiente Windows 9x/NT. Diferente do DynaLab, o PascalX apresenta um ambiente integrado para edição/compilação e execução/visualização.

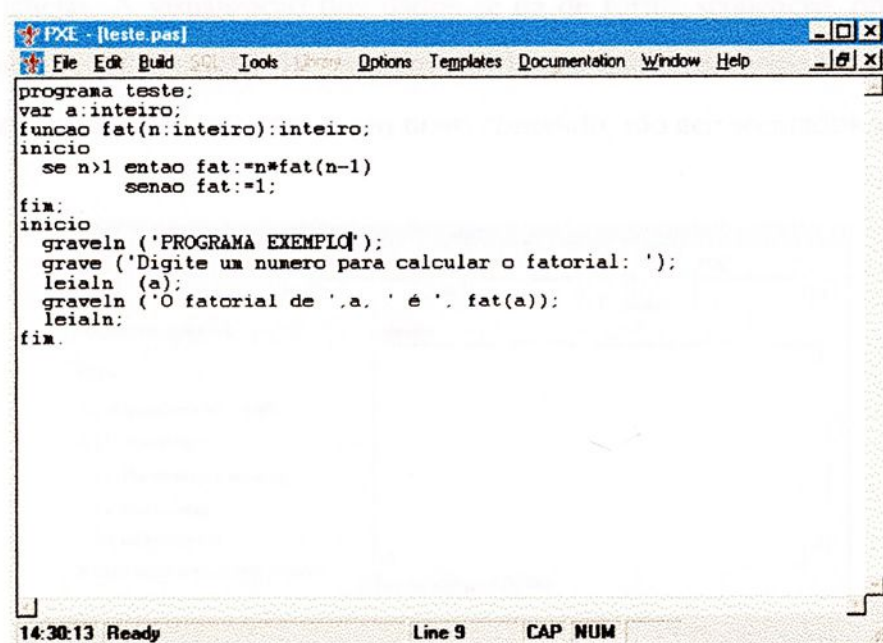


Figura 3.6 – Ambiente de desenvolvimento do PascalX

O ambiente de desenvolvimento é apresentado na figura 3.6. Ele oferece gabaritos para edição de comandos (veja figura 3.7), onde o usuário seleciona num menu a seção do código que deseja inserir – declaração de constantes, tipos, variáveis e/ou sub-rotinas; comandos de atribuição, seleção, repetição entre outros – e uma janela é exibida para sejam preenchidos alguns argumentos. Este tipo de recurso não força o usuário a

memorizar a sintaxe da linguagem. No entanto, só funciona para a linguagem Pascal. Uma ampla documentação do sistema e da gramática da linguagem fica disponível para o usuário através do menu *Help* (a interface da ferramenta é apresentada na língua inglesa).

Durante a compilação do programa, o usuário é auxiliado por mensagens precisas acusando o erro cometido e seu local (número da linha).

Uma vez compilado, a execução propriamente dita do programa se dá de modo direto em ambiente DOS (veja exemplo na figura 3.8). A visualização da execução é realizada posteriormente, considerando-se todas as entradas já realizadas pelo usuário até então, ou seja, o usuário não interage com novas entradas durante esta fase. Um ambiente de visualização (veja figura 3.9) é exibido ao usuário para que ele possa acompanhar a execução do programa passo a passo. Tanto a linha de código sendo executada, como o conteúdo dos dados sendo manipulados num determinado instante, são destacados nas respectivas janelas. A visualização dos dados se dá de forma sequencial, obedecendo ao fluxo de controle da execução do programa: toda vez que uma variável é atualizada, seu identificador, respectivamente com o seu novo conteúdo, são acrescentados ao fim da lista (janela).

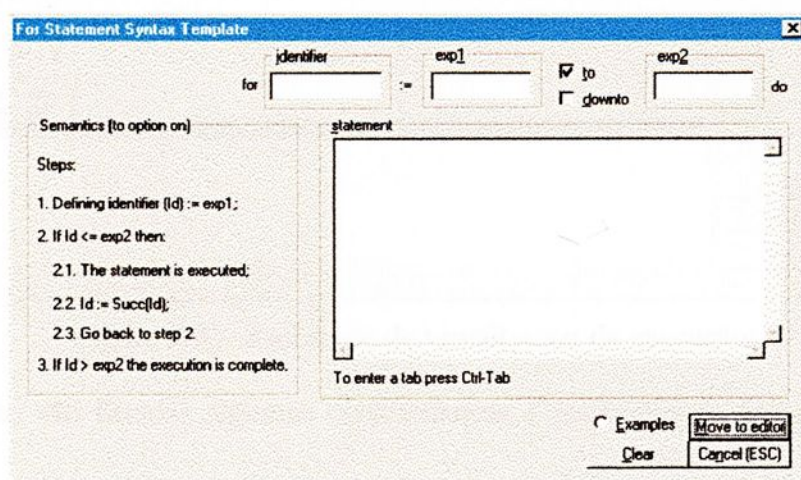


Figura 3.7 – Gabarito para edição de comandos

Um outro recurso bastante interessante apresentado pelo PascalX é a execução reversa (botão *Previous*); através dele podem-se desfazer alguns passos executados. Tal fato não força o estudante a reiniciar a execução do algoritmo caso um determinado passo não seja compreendido, basta apenas retroceder na execução e refazer os passos quantas vezes for necessário.

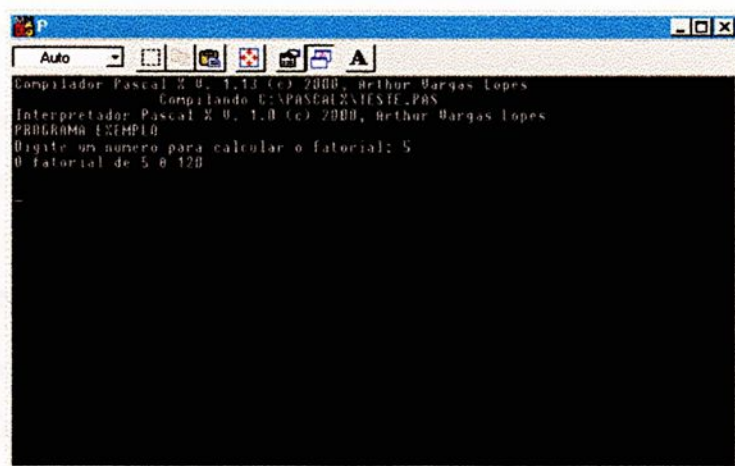


Figura 3.8 – Pré-execução em modo DOS

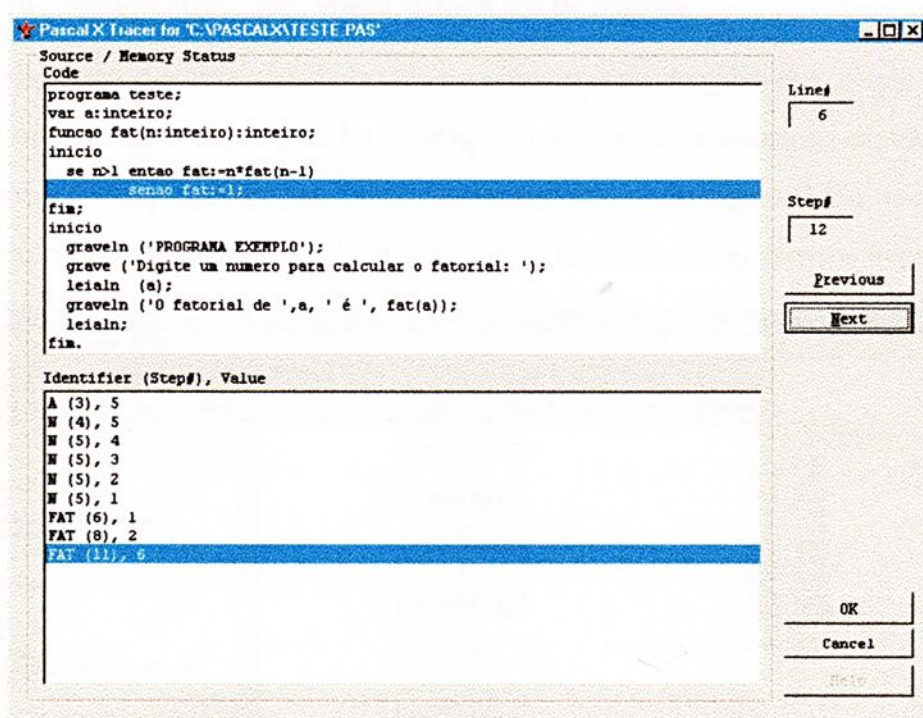


Figura 3.9 – Ambiente de visualização da execução

A seguir, são listadas algumas características e restrições da linguagem suportada pelo PascalX:

- Não suporta o uso de bibliotecas e funções externas;
- suporta vetor, registro e conjunto, bem como manipulação de arquivos;
- permite a implementação de sub-rotinas (*procedures* e *functions*) com passagens de parâmetro por valor e referência, suportando chamada recursiva;
- permite manipulação de ponteiros com alocação de variáveis dinâmicas.

3.2.5 Construtor

O software Construtor pode ser considerado uma ferramenta um tanto peculiar, se comparada as demais mencionadas. Com um conjunto de instruções reduzido, com sintaxe simples e a representação do programa feita através de diagramas de blocos, é uma opção bastante recomendada para acompanhar o estudante nos passos iniciais.

Desenvolvido pelo Centro Educacional de Informática Aplicada do SENAC (Rio de Janeiro), o Construtor opera sobre o sistema operacional Windows95/NT e suas versões subsequentes, oferecendo um ambiente de desenvolvimento de programas baseados em pseudocódigo, que utiliza termos em português.

A edição do algoritmo se dá exclusivamente através de gabaritos, o que desobriga o usuário de memorizar a sintaxe da linguagem. Ao selecionar a instrução desejada clicando sobre o ícone correspondente na barra de ferramentas (veja figura 3.10), um gabarito é exibido ao aprendiz (veja figura 3.11), que somente fornece os argumentos necessários.

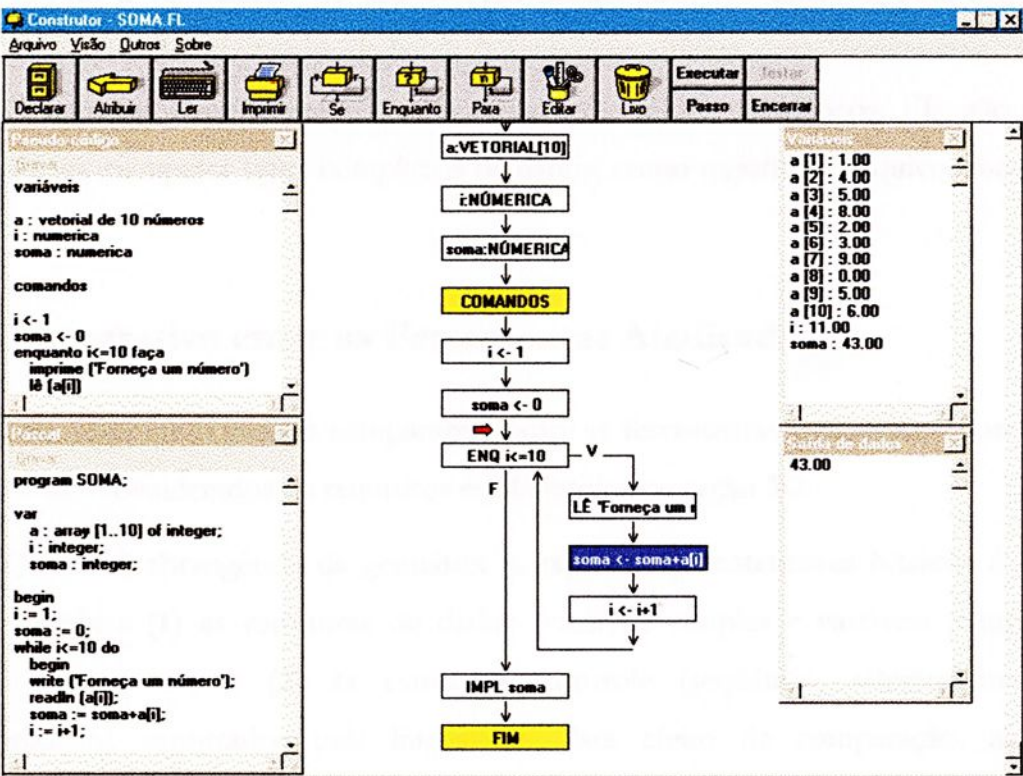


Figura 3.10 – Ambiente de desenvolvimento e visualização do Construtor

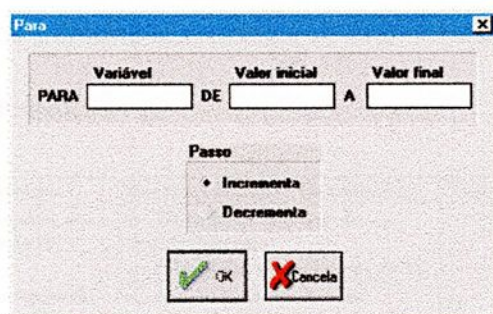


Figura 3.11 – Gabarito para edição de comandos

Diferente das demais ferramentas avaliadas, o algoritmo editado pelo usuário no Construtor é representado através de fluxograma. Esse aspecto facilita o acompanhamento do fluxo de controle durante a execução passo a passo.

Outro aspecto interessante oferecido pela ferramenta são as visões do programa que podem ser traduzidas para as linguagens Pascal, C e Clipper.

No início desta seção, foi afirmado que é uma ferramenta recomendada nos passos iniciais devido à valorização do seu aspecto pedagógico. No entanto, para avançar no ensino de construtores mais complexos é necessário utilizar outra ferramenta, uma vez que o Construtor implementa somente vetores de valores numéricos. Ele não oferece recursos para manipular tipos complexos de dados, como registros e arquivos, bem como sub-rotinas.

3.3 Comparativo entre as Ferramentas Analisadas

Para elaborar o quadro comparativo entre as ferramentas analisadas – apresentado a seguir – são considerados os requisitos estabelecidos na seção 3.2.

Quanto à abrangência da gramática, a expressão **construtores básicos** é utilizada para determinar (1) as estruturas de dados (variáveis simples e variáveis compostas – vetores e registros) e (2) as estruturas controle (seqüência, seleção, iteração e modularização) suportados pela linguagem. Para efeito de comparação, além dos construtores básicos, são considerados os itens a seguir:

- (1) Suporte a bibliotecas e funções externas;
- (2) Implementação de arquivos;

(3) Passagem de parâmetros e chamada recursiva de sub-rotinas

(4) Ponteiros e variáveis dinâmicas.

Ferramenta	Abrangência da Gramática	Suporte a Edição e Compilação	Visualização	Controle de Execução
ILA (pseudocódigo em português)	• dos construtores básicos, não permite a implementação de registros • suporta (3) com passagem de parâmetro somente por valor	• Não oferece ambiente de desenvolvimento • Compilação via prompt do DOS • tutorial na forma de texto descrevendo a sintaxe da linguagem com exemplos	-----	• Execução direta
Portugol/Plus (pseudocódigo em português baseado no PASCAL)	• todos os construtores básicos • suporta (3)	• ambiente de desenvolvimento com editor e compilador • não oferece ajuda sobre a sintaxe da linguagem	-----	• Execução direta
DynaLab (PASCAL)	• suporta todos os construtores básicos • suporta (3) • suporta (4)	• ambiente de desenvolvimento com editor e compilador • edição só por digitação direta	• Código/Dados (sincronizado)	• Execução direta • Execução reversa • Passo a passo
PascalX (pseudocódigo em português baseado em PASCAL e também PASCAL)	• suporta todos os construtores básicos • suporta (2) • suporta (3) • suporta (4)	• ambiente de desenvolvimento com editor e compilador • edição por digitação e gabaritos • tutorial com a gramática da linguagem na forma de hipertexto (em inglês da linguagem Pascal)	• Código/Dados (sincronizado)	• Execução direta • Execução reversa • Passo a passo
Construtor (pseudocódigo em português)	• dos construtores básicos só permite a implementação de vetor com valores inteiros, não suporta registro, não suporta sub-rotinas;	• ambiente de desenvolvimento com editor e compilador • edição baseada somente em gabaritos (auto-explicativo) • representação do algoritmo em diagrama de blocos	• Código/Dados (sincronizado)	• Execução direta • Passo a passo

Quadro 3.1 – Comparativo entre as ferramentas

A despeito de serem ferramentas já concebidas como produto, nota-se, através da avaliação feita, que apresentam algumas restrições ao se considerarem critérios de ergonomia e usabilidade de sua interface:

- o Portugol/Plus e o ILA, embora permitam a execução do algoritmo, não possuem um ambiente para visualizar este processo;
- o DynaLab não apresenta um ambiente integrado: o módulo de edição e compilação e o módulo de execução e visualização compõem dois aplicativos distintos que são executados de forma independente;
- o PascalX apresenta algumas restrições na visualização dos dados, e a execução passo a passo ocorre sobre uma execução prévia, não dando oportunidade de o usuário interagir com o algoritmo fornecendo novas entradas durante a fase de visualização;
- o Construtor, apesar de ser uma ferramenta didaticamente excelente por apresentar o modo de edição baseado em gabaritos e a representação do programa em diagramas de

blocos, peca na abrangência da gramática apresentada pela sua linguagem, que é bastante limitada, não permitindo a manipulação de alguns construtores importantes.

Tais fatos motivaram o desenvolvimento de uma ferramenta seguindo os mesmos princípios das supracitadas, porém tentando suprir algumas deficiências por elas apresentadas. Os detalhes da implementação desta ferramenta, assim como, a descrição de sua funcionalidade, são destacadas no capítulo a seguir.



Capítulo 4

AVA (Ambiente para Desenvolvimento e Visualização de Algoritmos)

O principal objetivo da ferramenta aqui descrita é dar suporte ao processo de ensino-aprendizagem de lógica de programação, oferecendo ao aluno um ambiente de fácil interação para a visualização da execução de seus algoritmos, estimulando-o na fase inicial dos cursos de Ciência da Computação e afins, onde o desenvolvimento da lógica de programação é de suma importância para o bom desempenho do aprendiz nas disciplinas subsequentes.

Batizada de AVA (Ambiente para Desenvolvimento e Visualização de Algoritmos), esta ferramenta adota uma linguagem baseada em pseudocódigo e permite ao usuário implementar e depurar seus algoritmos acompanhando sua execução passo a passo. A linguagem adotada segue o paradigma imperativo pelo fato deste ser o mais utilizado nas fases iniciais do ensino de programação (TIMÓTEO; BRASILEIRO, 1998).

Trata-se de um sistema de visualização de software que permite ao aluno analisar toda a dinâmica do funcionamento do algoritmo em questão; fato que seria extremamente difícil valendo-se do tradicional “teste de mesa” em que aluno executa o algoritmo mentalmente, rabiscando folhas de papel.

Neste capítulo são descritos detalhes referentes à implementação da ferramenta AVA. São abordados aspectos relacionados (1) aos requisitos do usuário que devem ser atendidos, (2) à linguagem algorítmica utilizada pelo usuário para codificar seus algoritmos, (3) ao processo de compilação e tradução do pseudocódigo numa estrutura que permita a sua execução e (4) à implementação do processo de visualização.

4.1 Especificação dos Requisitos

Um sistema de visualização de software que apresenta como princípio auxiliar o processo de ensino-aprendizagem deve permitir o seu uso tanto em sala de aula para exposição de conceitos pelo professor como em laboratório para a prática de resolução de problemas.

Para se conceber um sistema desse porte, deve-se levar em consideração as fases realizadas pelo “programador” para implementar uma solução a um dado problema (veja figura 4.1). Este processo obedece ao ciclo **descrição-execução-reflexão-depuração** estabelecido por Valente (1993b).

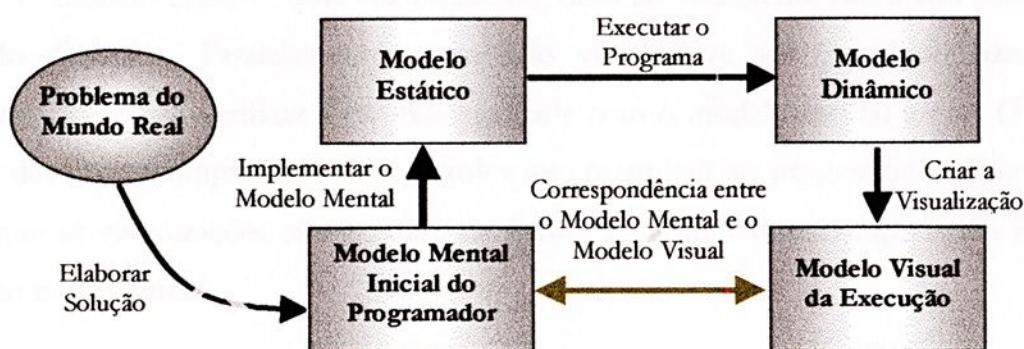


Figura 4.1 – Fases do processo de visualização de um algoritmo/programa

Partindo de um problema do mundo real, o programador (estudante) inicialmente cria o modelo mental do algoritmo/programa para solucioná-lo. Este modelo mental inicial é traduzido para o modelo estático através do uso de um editor de texto e de um compilador. Nesse contexto, o modelo estático compreende o código fonte, o código objeto associado e outras formas intermediárias de código. A partir do modelo estático, é gerado o modelo dinâmico (comportamento do programa em tempo de execução), valendo-se do sistema operacional e das bibliotecas de *run-time*. Este modelo deve ser mapeado de forma a representar o modelo mental inicial do programador (modelo visual da execução do programa). Este mapeamento possibilita ao programador (estudante) confrontar a execução do modelo dinâmico com o modelo mental que foi inicialmente concebido para solucionar um determinado problema. Tal fato torna possível compreender e depurar o programa mais facilmente. Quanto mais próximo o modelo visual da execução for do modelo mental, maior será a compreensão.

Partindo de um modelo mental do programador, um sistema de visualização de software deve atender de forma eficiente à concepção dos modelos estático, dinâmico e visual.

Para gerar o modelo estático, o sistema deve contar com uma linguagem que agregue construtores que suportem a implementação do modelo mental. O processo de edição e compilação também deve ser facilitado com recursos que minimizem o esforço do programador em memorizar a sintaxe da linguagem e em encontrar erros léxicos, sintáticos e semânticos cometidos durante a tradução do modelo mental para o modelo estático.

O modelo estático, uma vez finalizado, deve ser facilmente executado para gerar o modelo dinâmico. Paralelamente, o modelo visual deve ser concebido para que o programador possa verificar a sua fidedignidade com o modelo mental inicial. O modelo visual deve ser acompanhado de controles que permitam ao programador ditar o ritmo com que as visualizações são geradas, de forma a permitir uma compreensão maior da solução implementada.

4.2 Arquitetura do Sistema AVA

Com base nos requisitos estabelecidos na seção 4.1 chegou-se à elaboração da arquitetura do sistema AVA que é apresentada na figura 4.2.

O AVA é basicamente composto por dois módulos integrados:

- Ambiente de desenvolvimento
- Ambiente de visualização

O ambiente de desenvolvimento é composto por um simples editor de texto, onde o usuário pode editar o seu algoritmo ou abrir o arquivo fonte já editado para que possa ser compilado. No processo de edição, o esforço de memorização do conceito associado a um determinado construtor e da sua sintaxe é minimizado com o auxílio de um tutorial e gabaritos.

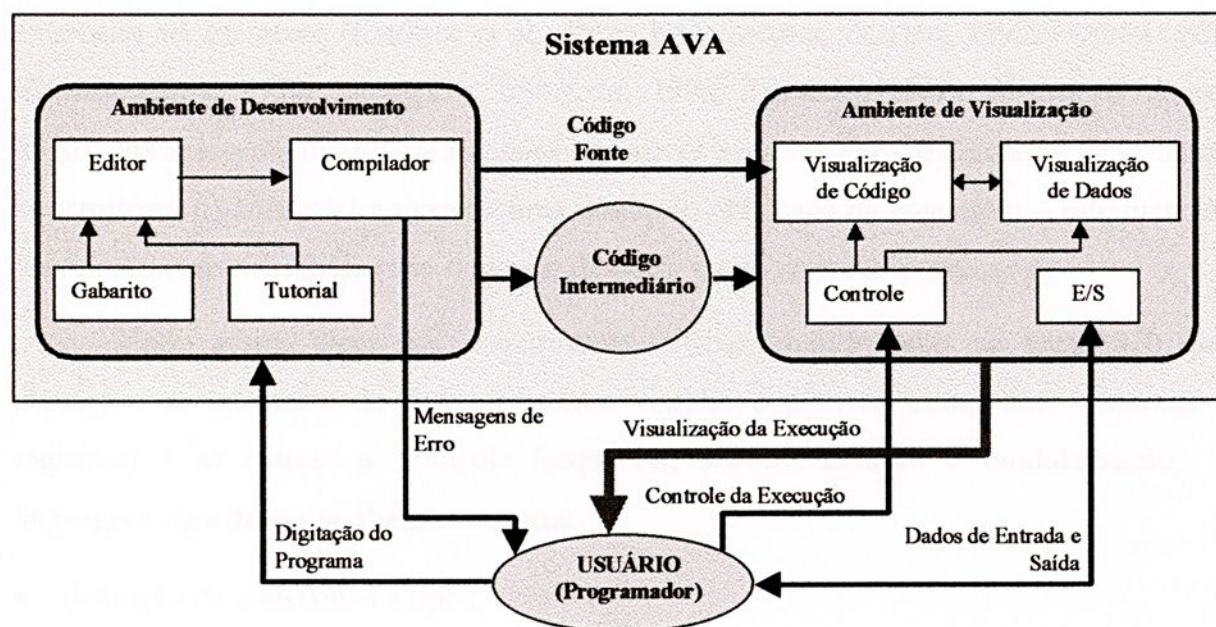


Figura 4.2 – Arquitetura do sistema AVA

Uma vez editado, o algoritmo passa por um compilador que acusa os erros léxicos, sintáticos e semânticos cometidos pelo usuário. Livre destes erros, o algoritmo é traduzido para um código intermediário para que possa ser executado.

O algoritmo é executado no ambiente de visualização, que conta com um visualizador de código (onde se pode acompanhar o fluxo da execução) e um visualizador de dados (que exibe o conteúdo dos dados manipulados pelo algoritmo durante a execução).

Através do painel de controle, o usuário estabelece o ritmo com que a execução do algoritmo se dá, podendo optar pela execução direta ou passo a passo.

Durante a execução do algoritmo, o usuário pode interagir com sua implementação fornecendo dados de entrada e recebendo as saídas geradas.

Nas seções subseqüentes, são abordados de forma detalhada os aspectos inerentes à funcionalidade do sistema e aos detalhes técnicos de sua implementação.

4.3 A Linguagem Algorítmica

A linguagem algorítmica adotada pelo AVA é baseada em pseudocódigo que utiliza termos em português, como tantas outras linguagens concebidas para fins didáticos

existentes na literatura (FARRER et al., 1999; FERNANDES; BOTINI, 1998; FOBELLONE; EBERSPACHER, 1999; SALVETTI; BARBOSA, 1998; TERADA, 1991). A maioria dessas linguagens apresenta diferenças mínimas na sintaxe e implementa basicamente os mesmos construtores (o anexo VI apresenta uma descrição detalhada da linguagem – este material foi inicialmente concebido com o intuito de servir como consulta aos alunos).

Sendo assim, além dos construtores básicos (mencionados na seção 3.3) que abrangem as estruturas de dados (variáveis simples e variáveis compostas – vetores e registros) e as estruturas controle (seqüência, seleção, iteração e modularização), a linguagem algorítmica também comporta:

- definição de constantes e tipos;
- rotinas (procedimentos e funções) com passagem de parâmetros por valor e referência;
- chamada recursiva de rotinas.

O programa a ser escrito na linguagem algorítmica deve apresentar a estrutura exibida no quadro 4.1:

```
algoritmo <identificador do algoritmo>  
    <declaração de constantes>  
    <declaração de tipos>  
    <declaração de variáveis>  
    <declaração de sub-rotinas>  
inicio  
    <bloco de comandos>  
finalgoritmo.
```

Quadro 4.1 – Estrutura do algoritmo

onde <identificador do algoritmo> é o nome do algoritmo, <declaração de constantes> é a zona de declaração de constantes, <declaração de tipos> é a zona de declaração de tipos, <declaração de variáveis> é a zona de declaração de variáveis, <declaração de sub-rotinas> é a zona de declaração de sub-rotinas, <bloco de comandos> é zona de especificação dos comandos a serem executados. No quadro 4.2, é apresentado o exemplo de um programa escrito na linguagem algorítmica. Este exemplo apresenta um algoritmo que calcula a média aritmética entre duas notas e exibe determinadas mensagens de acordo com o valor da média obtida.


```
Algoritmo Calcula_Media
variavel nota1,nota2,media:real;
inicio
  escreva ("Forneca as duas notas (0 a 10)");
  leia (nota1,nota2);
  media<-(nota1+nota2)/2;
  se (media>=7.0)
    entao escreva ("Aprovado!");
  senao se (media>=4.0)
    entao escreva ("De Exame!");
    senao escreva ("Reprovado!");
  fimse;
fimse;
fimalgoritmo.
```

Quadro 4.2 – Exemplo de um programa escrito na linguagem algorítmica

Logo após o cabeçalho do algoritmo o programador deve definir todos os objetos a serem utilizados (constantes, tipos, variáveis e sub-rotinas). Delimitados entre **inicio** e **fimalgoritmo**, devem ser especificados sequencialmente os comandos a serem executados. Estes envolvem comandos de entrada e saída, atribuição, estruturas de seleção, estruturas de iteração, chamadas a sub-rotinas entre outros (para maiores detalhes sobre a descrição da linguagem e sua sintaxe, veja anexo VI).

4.4 Módulo de Edição

O módulo de edição faz parte do ambiente de desenvolvimento do AVA (veja figura 4.3) e é composto por um simples editor de texto, onde o usuário pode iniciar a edição do seu algoritmo ou abrir o arquivo fonte já editado para que posteriormente possa ser compilado.

Este editor de texto conta com alguns recursos que são primários e padronizados em qualquer outro, como os menus **Arquivo**, **Editar** e **Localizar**.

Através do menu **Arquivo** o usuário pode optar por (1) abrir um algoritmo já editado (opção **Abrir**), (2) iniciar a edição de um novo algoritmo (opção **Novo**) – neste caso um novo documento já incluindo um código inicial como mostra do quadro 4.3 é criado automaticamente, (3) salvar o documento sendo editado ou salvar com um outro

nome (respectivamente opções **Salvar** e **Salvar Como**) e (4) encerrar a aplicação (opção **Sair**).

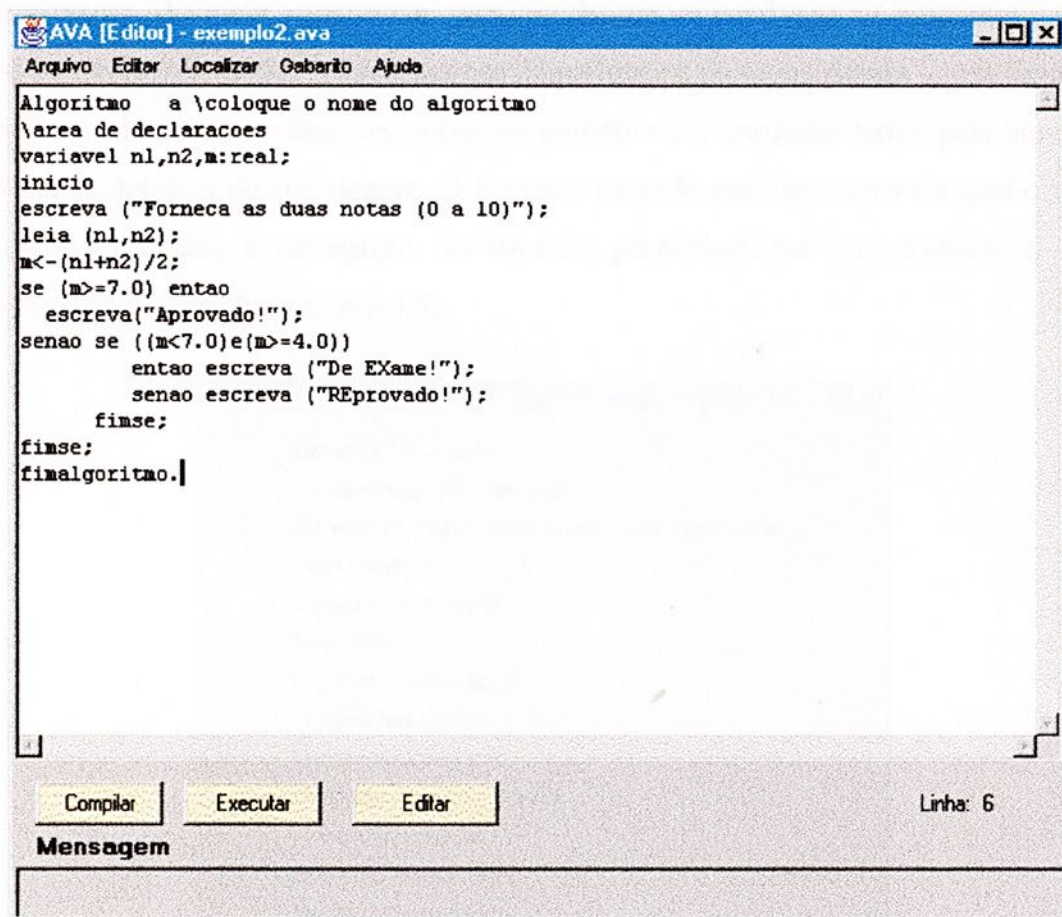


Figura 4.3 – Módulo de edição e compilação

```
algoritmo    \coloque o nome do algoritmo
\area de declaracoes
inicio
\area de comandos
finalgoritmo.
```

Quadro 4.3 – Código inicial gerado automaticamente ¹³

O menu **Editar** apresenta as opções **Recortar**, **Copiar**, **Colar** e **Selecionar Tudo**. E o **Localizar** as opções **Localizar**, **Localizar Novamente** e **Substituir**.

¹³ A partir deste código inicial, o usuário deve inserir aquilo lhe é solicitado pelos comentários (a linhas de comentários iniciam-se com o sinal “\”) e as demais instruções implementadas pelos construtores da linguagem.

Outros recursos mais importantes auxiliam o usuário no processo de edição visando a minimizar a sua necessidade em memorizar a sintaxe da linguagem e o funcionamento de cada construtor. Através de um tutorial que a ferramenta oferece (acessado através da opção **A Linguagem Algorítmica** do menu **Ajuda** – veja figura 4.4), o usuário pode obter explicações sobre os construtores implementados pela linguagem, assim como detalhes de sua sintaxe. O formato baseado em hipertexto no qual o tutorial está disposto facilita a navegação do usuário, permitindo-lhe que consulte o tópico desejado rapidamente (veja figura 4.5).

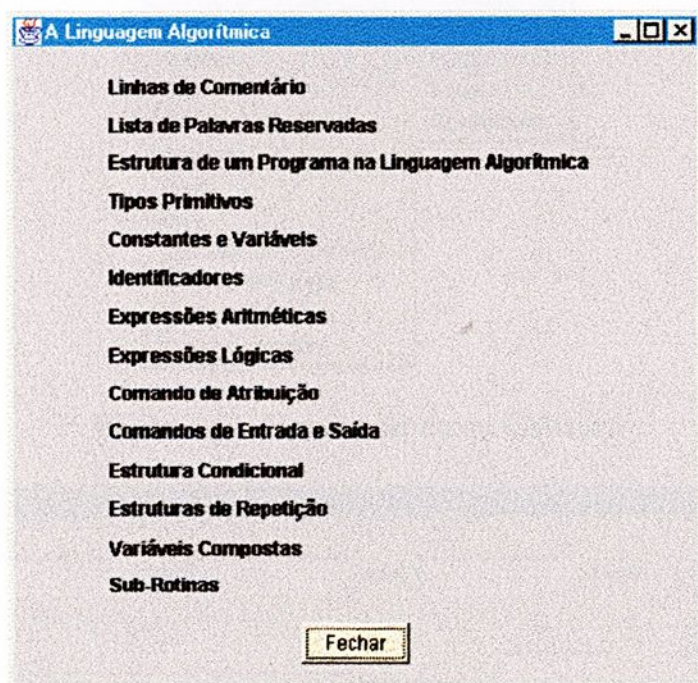


Figura 4.4 – Menu principal do tutorial

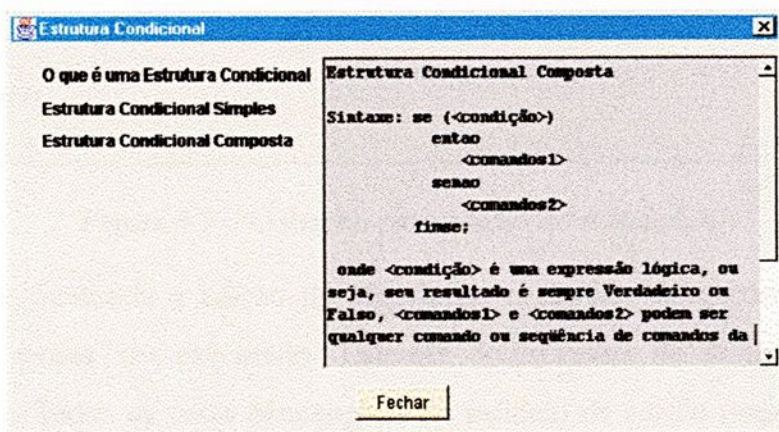


Figura 4.5 – Acesso ao tutorial

O usuário também pode se valer de gabaritos que o auxiliam na edição de qualquer um dos comandos comportados pela linguagem. Para tanto, deve acessar o menu **Gabarito** (veja figura 4.6) e selecionar o comando a ser inserido no texto. Dessa forma, uma janela de diálogo correspondente à opção escolhida é exibida ao usuário, que somente preenche os argumentos solicitados, sem se preocupar com os detalhes de sua sintaxe (veja figura 4.7). O texto referente à edição do comando em questão é inserido automaticamente no ponto de inserção determinado pela posição do cursor.

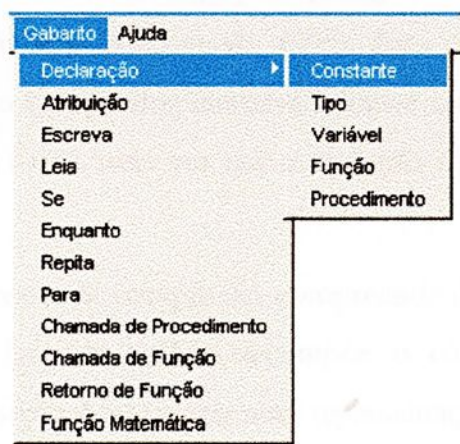


Figura 4.6 – Opções do menu Gabarito

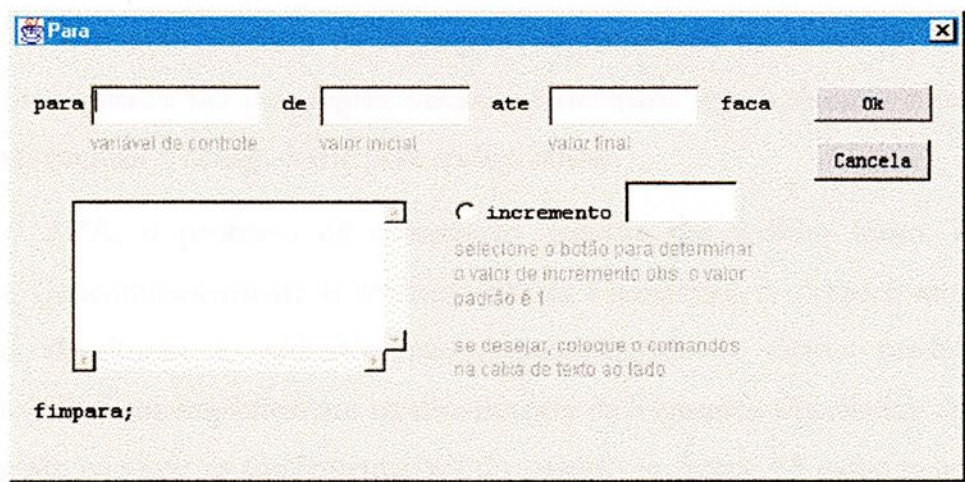


Figura 4.7 – Gabarito para edição de comandos

Uma vez finalizada a edição do algoritmo, o usuário deve compilá-lo para que posteriormente possa ser executado. Durante o processo de compilação, os erros detectados são exibidos na caixa **Mensagem** do módulo de edição (veja figura 4.3) onde são acusados a natureza do erro cometido e o local (número da linha) em que este

ocorreu. Maiores detalhes sobre a implementação do compilador são apresentados na seção a seguir.

4.5 O Compilador

Um compilador nada mais é que um programa que lê um outro programa escrito em uma linguagem de alto nível (código fonte) pelo programador e o traduz para um outro programa equivalente em outra linguagem (código objeto), a fim de que possa ser compreendido pelo computador (AHO et al., 1995). Esta é a sua função principal, no entanto, é importante que o compilador também informe ao programador a presença de erros cometidos no código fonte, uma vez que a tradução só deve ocorrer se este estiver livre de erros.

Sendo assim, o processo de compilação compreende duas fases: (1) fase de análise e (2) fase de síntese. A fase de análise decompõe o código fonte em suas partes constituintes (*tokens* – veja seção 4.5.1) e cria uma representação intermediária. Nesta fase, o compilador percorre o programa fonte coletando seus elementos constituintes e armazenando esta informação em uma estrutura hierárquica chamada árvore sintática (veja a árvore de derivação sintática da linguagem implementada no anexo II). Na fase de síntese, o compilador usa esta representação intermediária gerada durante o processo de análise para traduzir o código fonte para código objeto.

No AVA, o processo de compilação consiste nas análises léxica, sintática e semântica. Concomitantemente às análises sintática e semântica, o código fonte escrito na linguagem algorítmica é traduzido para representações de classes predefinidas na linguagem JAVA que implementam os construtores da linguagem algorítmica, dessa forma o AVA pode executar as implementações do usuário (a figura 4.8 exibe o processo de compilação do algoritmo). Nas seções a seguir, as análises léxica, sintática e semântica são apresentadas com detalhes.

4.5.1 Analisador Léxico

O analisador léxico é um componente do compilador que lê o programa fonte, caracter por caracter e constrói, a partir destas, entidades primárias chamadas *tokens*.



Basicamente, sua tarefa é identificar cada símbolo do programa fonte associando-o a um código predefinido (*token*).

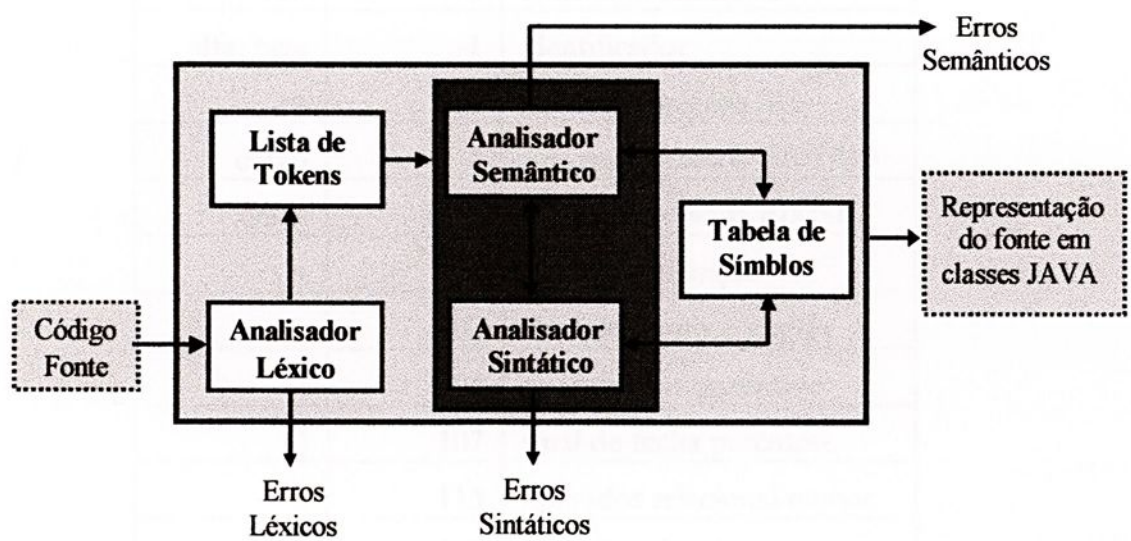


Figura 4.8 – Esquema de compilação de um algoritmo

Além de identificar os símbolos associando-os aos *tokens*, o analisador léxico também é responsável por:

- eliminar os espaços brancos, caracteres de fim de linha, tabulações;
- eliminar os comentários que acompanham o código fonte;
- informar o compilador sobre os erros que detectar.

O analisador léxico só acusa erros quando encontrar caracteres inválidos dentro de uma sequência, ou seja, um símbolo que não apresente um token correspondente válido (veja tabela de tokens válidos no anexo I).

O quadro 4.4 apresenta uma sentença (que constitui um exemplo de uma parte do código fonte de um algoritmo). O analisador léxico simplifica o texto de entrada em uma sequência de tokens (veja no quadro 4.5 a correspondência entre os símbolos reconhecidos e seus respectivos tokens).

```
se (alfa < 123)
    entao alfa <- alfa + beta;
fimse
```

Quadro 4.4 – Sentença de um programa fonte em linguagem algorítmica

Símbolos	Código do token	Descrição
123	-3	constante numérica
alfa, beta	-1	identificador
se	5	palavra reservada SE
entao	6	palavra reservada ENTAO
fimse	8	palavra reservada FIMSE
<-	100	sinal de atribuição
;	102	sinal de ponto e vírgula
(	106	sinal de abre parêntese
)	107	sinal de fecha parêntese
<	113	operador relacional menor
+	117	operador de adição

Quadro 4.5 – Tokens dos símbolos

Ao reconhecer os tokens durante a varredura de todo o programa fonte, o analisador léxico do AVA dispõe numa lista encadeada os símbolos reconhecidos com seus respectivos tokens. Dessa forma, para o exemplo do quadro 4.4, a lista ficaria como mostra a figura 4.9.

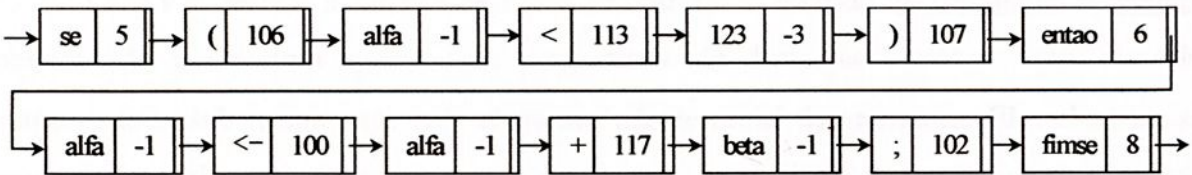


Figura 4.9 – Lista encadeada armazenando os tokens reconhecidos

Para o analisador léxico, a ordem em que os tokens estão dispostos não é relevante, basta que haja tokens correspondentes a todos os símbolos (seqüência de caracteres) do código fonte para que este esteja lexicamente correto. Um erro léxico só é acusado caso seja encontrada uma seqüência de caracteres para os quais não haja um token ao qual possa ser associado.

A partir da lista que é criada pelo analisador léxico, é possível ao compilador realizar a análise sintática e semântica. A partir deste momento, o programa fonte deixa de ser utilizado e as demais análises são feitas com base nos dados contidos nesta lista.

4.5.2 Analisador Sintático

Na análise sintática, os tokens agrupados na análise léxica são mapeados para uma estrutura chamada árvore de derivação sintática (veja a árvore de derivação sintática da linguagem algorítmica no anexo II) que expressa a ordem em que esses elementos devem ser avaliados. Para isso usa-se uma série de regras que constituem a gramática da linguagem algorítmica (veja a gramática da linguagem no anexo III).

É através dessa árvore que se estabelece, por exemplo, a precedência (ordem em que operações devem ser feitas, por exemplo, “*” antes de “+”) e a associatividade (ordem em que operações devem ser feitas, considerando a mesma precedência) dos operadores de uma expressão da linguagem. Se um token não puder ser “encaixado” nessa árvore sintática, então ocorre um erro sintático, significando que o programa fonte está incorreto, pois não está de acordo com as regras de formação de unidades sintáticas estabelecidas na gramática.

O analisador sintático, além de reconhecer a sequência de tokens e analisar sua estrutura, compila todos os seus atributos e os armazena na tabela de símbolos (veja seção 4.5.4).

O AVA realiza a análise sintática utilizando o método descendente recursivo, obedecendo à estrutura apresentada pela árvore de derivação sintática. Esse método emprega uma sub-rotina para cada elemento não-terminal da gramática. Elas chamam-se entre si, havendo recursão entre elas. Segundo Aho et al. (1995), uma gramática para esse tipo de analisador não pode ser ambígua, ou seja, não pode apresentar recursão à esquerda. Sendo assim, a gramática foi projetada para atender a esta restrição.

A figura 4.10 apresenta a mesma sentença do quadro 4.4 que é mapeado para uma árvore de derivação sintática. As regras da gramática que regem a formação desta árvore são apresentadas no quadro 4.6.

Ao realizar a varredura dessa sentença (que já foi analisada lexicamente e seus tokens foram incluídos numa lista), o analisador sintático, ao reconhecer o token de código “5”, saberá que se trata de uma instrução condicional e tentará gerar uma árvore

sintática onde todos os demais tokens da lista possam ser consumidos de acordo a gramática estabelecida.

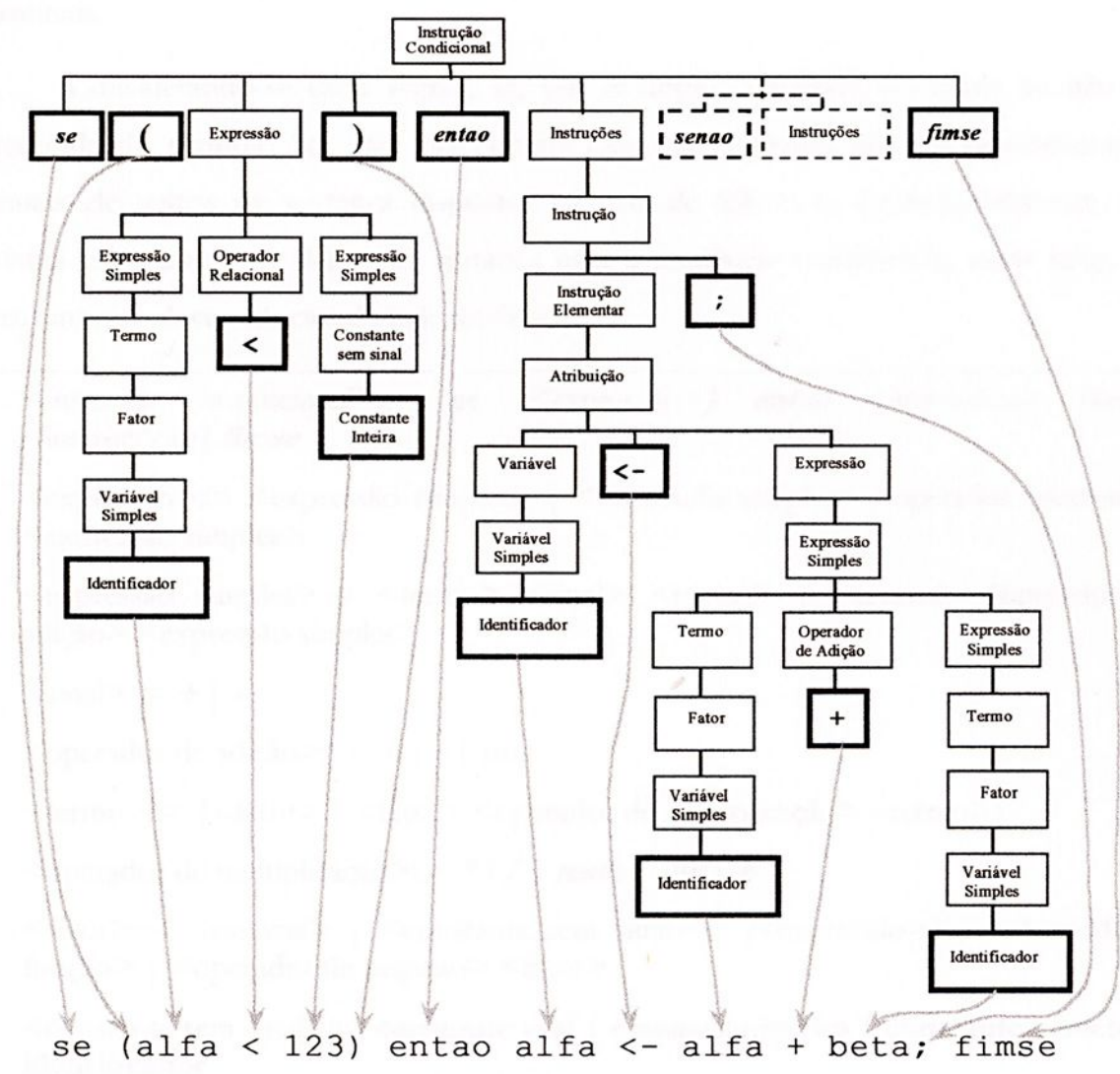


Figura 4.10 – Árvore de derivação sintática

Neste exemplo, a regra a ser aplicada é a que reconhece uma instrução condicional (regra <instrução condicional>); esta regra determina que o primeiro token deve identificar a palavra reservada **se** seguido do símbolo “(”; logo a seguir uma expressão deve ser reconhecida (regra <expressão>); o símbolo “)” deve vir logo a seguir; na seqüência deve vir a palavra reservada **entao** seguido por instruções (que são reconhecidas pela regra <instruções>); a seqüência formada pela palavra reservada **senao** e por <instruções> apresenta-se como opcional, ou seja, pode ser omitida; e o último símbolo a ser reconhecido na instrução deve ser a palavra reservada **fimse**. Cada uma das regras é derivada em elementos terminais (neste exemplo, os elementos são **se**, **(**, **)**, **entao**, **senao**,

fimse) e elementos não terminais, que correspondem a outras regras da gramática (<expressão> e <instruções>), e se derivam sucessivamente até chegar aos elementos terminais.

Considerando-se estas regras, se, por exemplo, a palavra reservada **se** não for precedida do símbolo “(” não haverá uma regra da gramática que possa consumir os demais elementos da sentença dispostos na lista de tokens e, conseqüentemente, não existirá uma árvore de derivação sintática para a sequência considerada; neste caso, um erro sintático deve ser acusado pelo analisador.

- <instrução condicional> ::= **se** (<expressão>) **entao** <instruções> [**senao** <instruções>] **fimse**
- <expressão> ::= <expressão simples> | <expressão simples> <operador relacional> <expressão simples>
- <expressão simples> ::= <termo> <sinal> <termo> | <termo> <operador de adição> <expressão simples>
- <sinal> ::= + | -
- <operador de adição> ::= + | - | **ou**
- <termo> ::= { <fator> | <fator> <operador de multiplicação> <termo> }
- <operador de multiplicação> ::= * | / | **resto** | **div** | **e**
- <fator> ::= <variável> | <constante sem sinal> | (<expressão>) | <chamada de função> | <operador de negação> <fator>
- <constante sem sinal> ::= **constante real** | **constante inteira** | **constante caracter** | **identificador**
- <operador relacional> ::= = | > | < | >= | <= | <>
- <variável> ::= <variável simples> | <variável composta>
- <variável simples> ::= **identificador**
- <instruções> ::= <instrução> | <instrução> <instruções>
- <instrução> ::= { <instrução elemental> | <instrução estruturada> } ;
- <instrução elemental> ::= <atribuição> | <leitura> | <escrita> | <chamada de procedimento>
- <atribuição> ::= <variável> <-> <expressão>

Quadro 4.6 – Regras de derivação utilizadas para gerar a árvore sintática da figura 4.10

4.5.3 Analisador Semântico

O analisador semântico se preocupa em analisar se a sentença (seqüência formada pelos elementos terminais já analisada sintaticamente) tem algum significado. Podem ser encontradas sentenças que sintaticamente estão corretas, mas não semanticamente, pois podem não apresentar sentido.

No AVA a análise semântica é realizada simultaneamente à análise sintática, ou seja, é verificado se os elementos terminais estão dispostos corretamente, e também é analisado se os mesmos apresentam algum sentido.

Basicamente, a análise semântica constitui-se em um verificador de tipos. Este analisador é responsável por verificar (1) se um operador contém operandos do mesmo tipo, (2) se estruturas condicionais (instrução **se**) e estruturas de repetição condicionadas (**repita** e **enquanto**) possuem expressões lógicas, (3) se um identificador referenciado no código fonte foi declarado, entre outras atribuições. Se alguma dessas condições não for satisfeita, o sistema reportará um erro semântico.

Tomando-se como exemplo a mesma sentença apresentada no quadro 4.4, a análise semântica se processa como segue:

`se (alfa < 123) entao alfa <- alfa + beta; fimse`

1. Na expressão entre parênteses após a palavra reservada **se**, faz-se necessária uma expressão cujo valor resultante é lógico (verdadeiro ou falso);
2. A presença de um operador relacional (sinal de menor – “<”) obriga a presença de dois operandos do mesmo tipo de cada um dos lados deste operador;
3. A instrução que segue a palavra reservada **entao**, sendo uma atribuição, obriga que a variável que aparece do lado esquerdo do sinal (“<-”) seja do mesmo tipo que o resultado da expressão do lado direito;
4. A cada identificador encontrado na sentença (**alfa** e **beta**), é necessário verificar se o mesmo foi declarado.

Logicamente, para que a análise semântica seja realizada, todos os dados referentes aos identificadores devem estar registrados na tabela de símbolos (veja seção 4.5.4).

Concomitantemente à verificação da análise semântica, o compilador do AVA vai convertendo o código fonte para uma estrutura dinâmica composta por classes predefinidas na linguagem JAVA, que permitirá a sua execução no ambiente (veja detalhes sobre a estrutura dinâmica na seção a seguir).

4.5.4 Tabela de Símbolos / Estrutura Dinâmica para Representação do Algoritmo

Um compilador necessita coletar e usar informações sobre os nomes dos objetos (identificadores) que aparecem no programa fonte. Estas informações são colocadas numa estrutura de dados chamada **Tabela de Símbolos**. As informações coletadas sobre os identificadores incluem: a cadeia de caracteres pelo qual é nomeado, isto é, o nome propriamente dito; seu tipo - inteiro, real, lógico etc; sua forma - variável simples, composta; seu tamanho; sua localização na memória; e outros atributos, dependendo da linguagem (AHO et al., 1995). Geralmente, estas informações são introduzidas na tabela durante as análises léxica e sintática.

As informações contidas na tabela de símbolos são usadas em diversas fases do processo de compilação. Na análise semântica, a tabela é acessada para verificar se o uso de um identificador é consistente com sua declaração. Por exemplo, numa expressão aritmética ou lógica, as variáveis devem ser compatíveis entre si. Também na fase de geração de código, tais informações são utilizadas para saber quanto e que tipo de memória em tempo de execução deve ser alocado para determinado identificador.

No compilador do AVA, essa tabela faz parte de uma estrutura dinâmica que permite a execução do algoritmo. Esta estrutura, além dos dados dos identificadores, também incorpora a sequência de instruções que deve ser executada, sendo que a cada instrução são associados os objetos que estão envolvidos.

Essa estrutura é criada a partir da análise sintática no momento em que é feita a varredura da zona de declarações (constantes, tipos, variáveis, sub-rotinas, parâmetros).

Todos os objetos declarados são incluídos nessa estrutura, juntamente com os dados que os caracterizam (a figura 4.11 apresenta uma visão parcial da estrutura dinâmica). Estes dados são utilizados posteriormente, durante o curso das análises sintática e semântica, para realizar a varredura na zona de instruções.

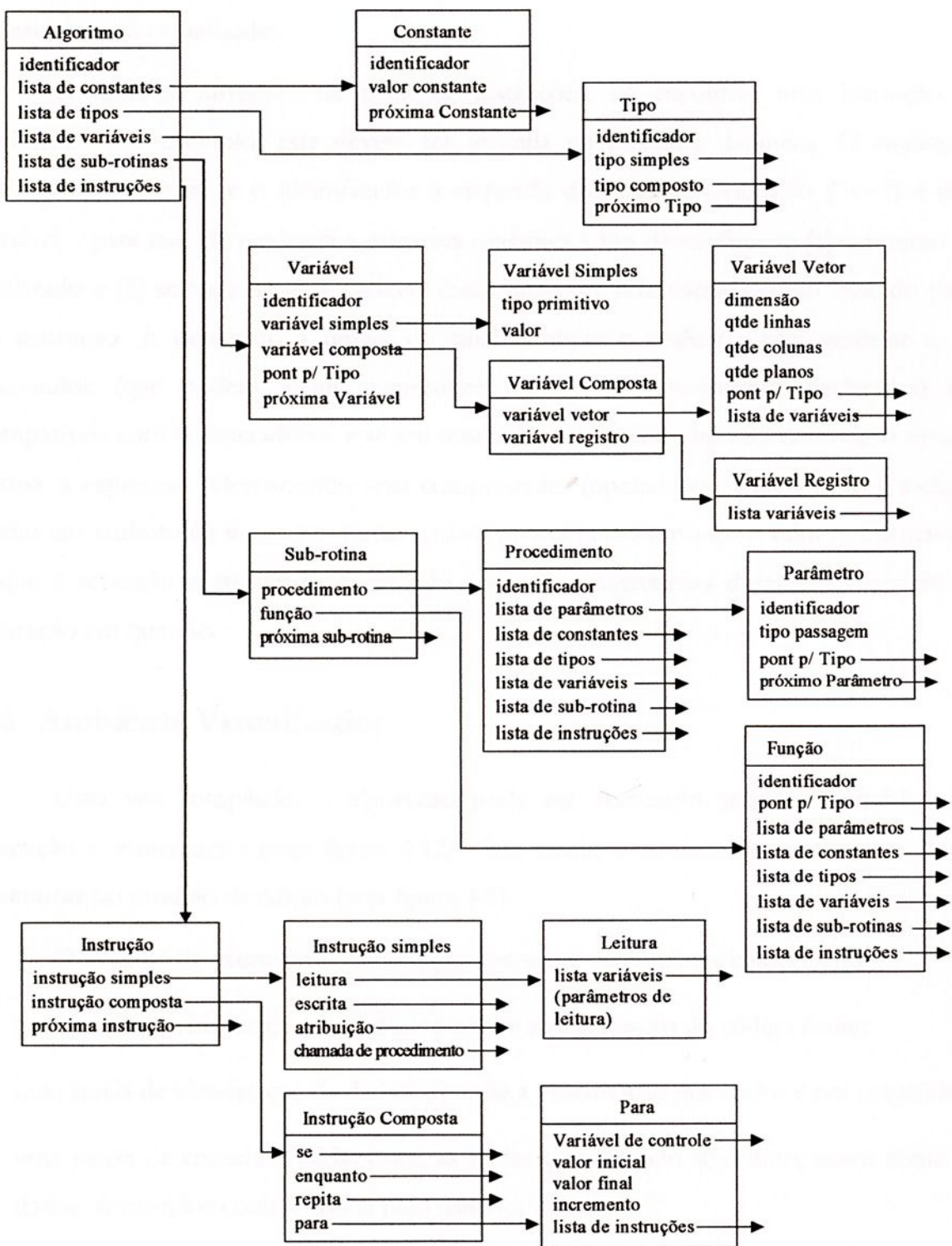


Figura 4.11 – Estrutura dinâmica (parcial) que permite a execução do algoritmo

Neste contexto, o analisador sintático, ao encontrar o símbolo **variavel**, por exemplo, incluirá na lista de variáveis do algoritmo (representado pela estrutura dinâmica na figura 4.11) todos os identificadores que se seguem, sabendo que a especificação do tipo deverá ser feita ao final da lista de identificadores, momento em que todas as variáveis da estrutura são atualizadas.

Durante a varredura na zona de instruções, ao encontrar uma instrução de atribuição, por exemplo, esta deverá ser incluída na estrutura dinâmica. O analisador semântico verificará se o identificador à esquerda do sinal de atribuição (“<-”) é uma variável – para isso ele recorrerá à estrutura dinâmica a fim de verificar se (1) o mesmo foi declarado e (2) se trata de uma variável. Esta variável é referenciada como fazendo parte da instrução. A expressão à direita do sinal também é analisada para verificar se os operandos (que podem incluir constantes e variáveis previamente declaradas) são compatíveis com os operadores, e se seu resultado é do mesmo tipo da variável; da mesma forma, a expressão referenciando seus componentes (operadores e operandos) é incluída como um atributo da instrução. Cada variável possui como atributo o valor que armazena e que é acessado e atualizado de acordo com as circunstâncias durante a execução da instrução em questão.

4.6 Ambiente Visualizador

Uma vez compilado, o algoritmo pode ser executado através do módulo de execução e visualização (veja figura 4.12). Para tanto, o usuário deve acionar o botão **Executar** no módulo de edição (veja figura 4.3).

O módulo de execução e visualização apresenta os seguintes componentes:

- uma janela de visualização de código: permite a visualização do código fonte;
- uma janela de visualização de dados: permite a visualização dos dados e seu conteúdo;
- uma janela de entrada e saída: exibe as saídas geradas pelo algoritmo, assim como os dados fornecidos como entrada pelo usuário;
- painel de controle: controla o ritmo de execução do algoritmo;

- uma janela de mensagens: exibe mensagens de erro de execução (divisão por zero, “overflow” do índice do vetor, entre outros).

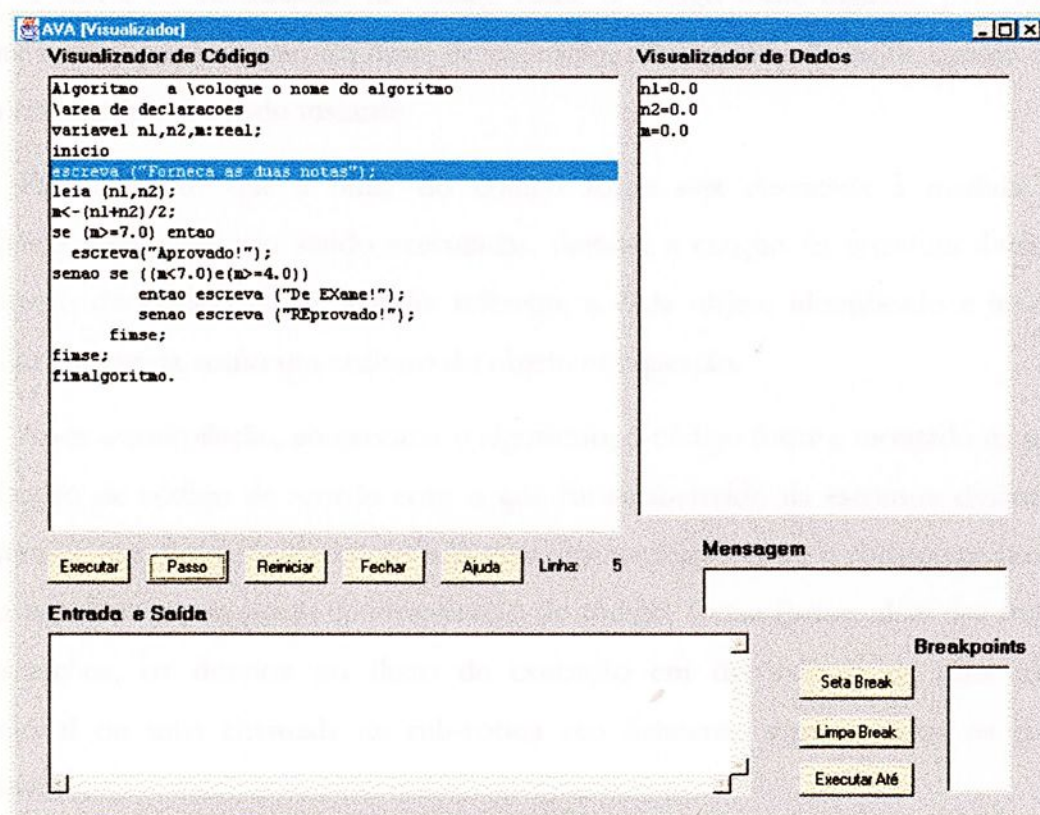


Figura 4.12 – Ambiente de Execução e Visualização

Através do painel de controle, o usuário pode optar pela execução direta (acionando o botão **Executar**) ou pela execução passo a passo (acionando o botão **Passo**). Na primeira opção, o algoritmo é executado do começo ao fim, enquanto na segunda, o botão **Passo** deve ser acionado para a execução de cada comando.

Através deste módulo, o aprendiz pode acompanhar o fluxo de execução do algoritmo passo a passo, visualizando o código em execução na janela de visualização de código e os dados sendo manipulados na janela de visualização de dados – a cada comando executado, as linhas referentes ao código e aos dados (constantes, variáveis e parâmetros) são destacadas nas respectivas janelas. Dessa forma, na chamada de uma sub-rotina (função ou procedimento) os dados locais são exibidos e posteriormente removidos após a sua execução, enquanto o código referente à sua implementação também é destacado, deixando claro o desvio no fluxo de execução.

4.6.1 Visualização do Código

A janela de visualização de código exibe o código fonte digitado pelo usuário e permite o acompanhamento do fluxo de execução, destacando a linha de código que está sendo executada num dado instante.

Para permitir que a linha do código fonte seja destacada à medida que as respectivas instruções vão sendo executadas, durante a criação da estrutura dinâmica na compilação do código fonte, a linha referente a cada objeto identificado e inserida na estrutura é mantida como um atributo do objeto em questão.

Após a compilação, ao executar o algoritmo, o código fonte é montado na janela de visualização de código de acordo com o que foi estabelecido na estrutura dinâmica. Tal fato permite que durante a execução haja uma sincronização entre o código em execução e a linha em destaque na janela de visualização de código. Dessa forma, além das seqüências de instruções, os desvios no fluxo de execução em decorrência de uma instrução condicional ou uma chamada de sub-rotina são fielmente representados na janela de visualização.

4.6.2 Visualização dos Dados

A janela de visualização de dados permite visualizar os objetos do algoritmo responsáveis por armazenar dados (constantes, variáveis e parâmetros).

Durante a execução do código referente à declaração desses objetos, seu identificador é inserido numa lista (janela de visualização de dados) e o número da linha referente à posição na lista é armazenado como atributo desse objeto.

A representação desses objetos se dá na sua forma primitiva: inteiro, real, caracter e lógico. Caso haja um objeto (variável ou parâmetro) que seja composto ele é decomposto e cada componente é inserido numa posição da lista. Por exemplo, suponha declaração de uma variável como mostra o quadro 4.7:

```
tipo v=vetor[5] de inteiro;  
variavel a: v;
```

Quadro 4.7 – Declaração de uma variável composta

Neste caso, a variável será representada como mostra o quadro 4.8:

a [1] = 0
a [2] = 0
a [3] = 0
a [4] = 0
a [5] = 0

Quadro 4.8 – Representação de um vetor na janela de visualização de dados

Como os objetos são inseridos na lista quando a sua declaração é executada, numa chamada de sub-rotina (procedimento ou função), novos elementos também são inseridos na lista (parâmetros e variáveis locais). Uma vez que se encerre a execução da sub-rotina, esses elementos são removidos.

Todas as variáveis, ao serem inseridas na lista, são exibidas com o seu valor inicial:

- variável inteira = 0;
- variável real = 0.0;
- variável caracter = “”;
- variável lógica = VERDADEIRO.

A execução de uma instrução que altere o valor de uma variável é representada com a sua respectiva atualização na lista.

4.6.3 Janela de Entrada e Saída

A janela de entrada e saída exibe os dados gerados pela execução do algoritmo através de uma instrução de saída (**escreva**), e também os dados fornecidos pelo usuário em resposta à execução de uma instrução de entrada (**leia**). Esses dados são exibidos sequencialmente uma linha após a outra.

A saída gerada pelo algoritmo serve como uma interface exibindo mensagens para solicitar dados de entrada e resultados ao usuário.

A execução de uma instrução de entrada interrompe o processamento para que o usuário possa fornecer um dado. Uma janela é exibida (veja figura 4.13) e após a digitação

da entrada, esta é incluída na janela de entrada e saída para manter um histórico dos valores fornecidos.



Figura 4.13 – Janela de leitura

4.6.4 Controle da Execução do Algoritmo

O usuário pode controlar o ritmo de execução do algoritmo através dos botões **Executar**, **Passo**, **Reiniciar**.

Ao acionar o botão **Executar**, independente do momento, o algoritmo é executado até o seu final.

O botão **Passo** determina a execução do algoritmo linha por linha (execução passo a passo); neste caso, o botão deve ser acionado para a execução de cada instrução.

Através do botão **Reiniciar**, é possível abortar qualquer execução para reiniciar o processo.

Durante a execução passo a passo o usuário pode estabelecer *breakpoints* para determinar pontos de parada durante a execução. Para tanto, ele deve selecionar uma linha de código na janela de visualização de código e acionar o botão **Seta Break**; o número referente à linha é inserido na janela de breakpoints; podem ser inseridos quantos breakpoints forem desejados.

Para fazer uso do breakpoint, o usuário deve acionar o botão **Executar Até**; dessa forma, a execução, que até o momento se processava passo a passo, alterna para a execução direta até encontrar uma linha que esteja estabelecida como breakpoint. A partir daí, o usuário pode retomar a execução passo a passo ou executar direto até o próximo breakpoint; caso nenhuma das linhas estabelecidas como breakpoint faça parte do fluxo de execução do algoritmo, este é executado até o seu final.

Para remover um breakpoint, basta selecioná-lo na sua respectiva janela e acionar o botão **Remover**.

4.7 Características, Limitações e Deficiências da Ferramenta

O hardware utilizado para o desenvolvimento da ferramenta é baseado num IBM-PC ou compatível, pelo fato de ser uma plataforma amplamente utilizada e em quantidade suficiente no meio acadêmico. A implementação foi feita numa linguagem de programação orientada a objeto, mais precisamente Java, pelo fato desta ser facilmente portátil para várias plataformas de hardware e software. O ambiente de desenvolvimento adotado foi o Visual Cafe¹⁴ sobre o sistema operacional Windows95 (ou suas versões posteriores).

O AVA comporta estruturas de dados (variáveis simples e compostas – vetores e registros) e estruturas de controle (seqüência, seleção, iteração e modularização): uma variável **vetor** pode ter até três dimensões e podem ser combinados vetores com registros recursivamente; os módulos (**funções** e **procedimentos**) podem ser implementados com passagem de parâmetros (por **valor** e **referência**) e chamadas recursivas.

Algumas limitações estão relacionadas à utilização de bibliotecas e funções externas, e implementação de arquivos e manipulação de ponteiros e variáveis dinâmicas que inicialmente não foram implementados na linguagem adotada.

Quanto à funcionalidade, um recurso de grande valia que não é oferecido pelo AVA é a execução reversa (está prevista a incorporação deste recurso futuramente). Aspectos relacionados à criação de animação que represente a execução do algoritmo (por exemplo, figuras representando a inserção de um nó numa árvore binária) também não são oferecidos nem estão previstos.

Um aspecto que pode ser aperfeiçoado diz respeito às mensagens de erros. Inicialmente foi estabelecido um conjunto abrangendo os erros mais comuns de sintaxe e semântica, no entanto, através do experimento pôde-se constatar que uma série de erros cometidos pelos alunos não foram previstos. Isso causou uma série de inconvenientes devido ao fato de o erro reportado não condizer com o que realmente foi cometido, desviando atenção a aluno. Num próximo passo pretende-se catalogar os erros mais comuns (que ainda não foram tratados) e incorporá-los à ferramenta.

¹⁴ Visual Cafe for JAVA Database Development Edition 2.5 (Symantec Corporation)



Alguns aspectos no tocante à ergonomia da interface também podem ser aperfeiçoados, de forma a melhorar a interação com o usuário, como, por exemplo, na janela de leitura, em que o usuário é obrigado a acionar o botão **OK** para efetivar a entrada de dados. O redimensionamento dinâmico das janelas de visualização deixando a cargo do usuário estabelecer o tamanho da área desejada para visualizar o algoritmo também seria um aspecto desejável. As janelas de consulta ao tutorial podem ser melhoradas de forma a ampliar o raio de visão do usuário para tópicos e exemplos muito extensos; também podem ser incluídas funções de acesso baseadas em busca por palavras-chave.

Com o desenvolvimento da ferramenta AVA, buscou-se dar uma abordagem alternativa ao ensino de algoritmos, visando a estimular os alunos a participarem mais ativamente das aulas, além de servir como auxílio no desenvolvimento de atividades extra-aula. A experiência da aplicação da ferramenta no processo de ensino-aprendizagem em lógica de programação é relatada no capítulo a seguir.



Capítulo 5

Caracterização do Ensino de Lógica de Programação nos Cursos Superiores de Informática

Azeredo (2000) afirma que o ensino da primeira linguagem de programação deve ser precedido do estudo de algoritmos, ou seja, o aprendiz deve, inicialmente, ser capaz de elaborar soluções antes de programá-los em uma determinada linguagem executável.

Sendo assim, dentre as matérias que compõem o currículo dos cursos de informática, uma atenção especial deve ser dedicada à lógica de programação, pelo fato desta servir de base para todas as disciplinas que envolvem programação. De acordo com as Diretrizes Curriculares (MEC, 1999):

... programação, entendida como programação de computadores, é uma atividade voltada à solução de problemas. Nesse sentido ela está relacionada com uma variada gama de outras atividades como especificação, projeto, validação, modelagem e estruturação de programas e dados, utilizando-se das linguagens de programação propriamente ditas, como ferramentas...

A lógica de programação é o ponto inicial para inserir o estudante no mundo da informática, desenvolvendo nele a habilidade de analisar um problema e elaborar uma solução que seja viável do ponto de vista computacional.

O estudo apresentado neste capítulo inicia-se através de uma reflexão sobre o ensino da lógica de programação, obedecendo aos preceitos das diretrizes curriculares, contrastando-os com a prática aplicada em diversos cursos de informática oferecidos por diferentes instituições, e finda com o relato de uma experiência realizada na Faculdade de Ciências Exatas e Tecnológicas da Universidade de Marília.

5.1 O Ensino da Lógica de Programação

De acordo com as Diretrizes Curriculares¹⁵ (MEC, 1998) dos cursos de graduação voltados à área de informática estabelecidas pela Comissão de Especialistas de Ensino de Computação e Informática (CEEInf), que abrangem os cursos de Sistemas de Informação, Ciência da Computação, Licenciatura em Computação e Engenharia da Computação, o ensino da (lógica de) programação faz parte do rol das disciplinas da área de formação básica da ciência da computação (compreende o núcleo de matérias que envolvem a parte científica e as técnicas fundamentais à formação sólida dos egressos dos diversos cursos de computação). Denominada **técnicas de programação**, no Currículo de Referência¹⁶ da Sociedade Brasileira de Computação (SBC, 1999), esta disciplina abrange os seguintes tópicos:

- Desenvolvimento de algoritmos
- Tipos de dados básicos e estruturados
- Comandos de uma linguagem de programação
- Metodologia de desenvolvimento de programas
- Modularidade e abstração

Da forma como foi estabelecida, esta disciplina não se restringe ao estudo de linguagens de programação. Estas constituem-se em uma ferramenta de concepção do

¹⁵ O MEC estabeleceu uma estrutura curricular abstrata para compor as Diretrizes Curriculares dos Cursos de Graduação. Essas diretrizes são organizadas de tal forma que as Instituições de Ensino Superior possam, a partir dela, exercer a criatividade e conceber currículos plenos diversificados. Esta estrutura abstrata pode ser vista como uma “especificação de requisitos” que serve de ponto inicial para que, através de um processo de detalhamentos sucessivos, possa se chegar a uma rede de disciplinas distribuídas no tempo, estabelecendo, dessa forma, o currículo pleno a ser executado por um corpo de professores. A relação das diretrizes curriculares de diversos cursos de graduação pode ser obtida através da Internet no endereço <http://www.mec.gov.br/Sesu/diretriz.shtm>.

¹⁶ De acordo com Azeredo (2000), a disciplina caracterizada como *Programação* nas Diretrizes Curriculares do MEC, consta no Currículo de Referência da SBC como parte do núcleo de Fundamentos de Computação e abrange quatro matérias: Linguagens de Programação, Técnicas de Programação, Estrutura de Dados e Pesquisa e Ordenação. Apesar das diferentes denominações e divisões, o autor considera que os conteúdos de ambas as propostas se equivalem.

software, que representa o resultado da aplicação de uma série de conhecimentos que transformam a especificação da solução de um problema em um programa de computador que efetivamente resolve aquele problema (MEC, 1998). Notadamente, o desenvolvimento de algoritmos, juntamente com o estudo de estruturas de dados, deve receber especial atenção na abordagem do tema programação.

Como afirmado anteriormente, o contato com a primeira linguagem de programação executável deve ser precedido pelo estudo de algoritmos (AZEREDO, 2000), onde conceitos como variáveis, constantes, operadores, expressões aritméticas e estruturas de controle (atribuição, seqüência, seleção, repetição, recursão) devem ser amplamente explorados e, também pelo estudo dos principais paradigmas de programação - imperativa, funcional, lógica e orientada a objetos (MEC, 1998).

O estudo inicial de algoritmos, mais precisamente o projeto e a construção de algoritmos, justifica-se pelo fato de ele servir como base para se elaborar um programa de computador; ele é a representação da solução de um problema. Segundo Knuth (1997), um algoritmo pode ser definido como uma seqüência finita de instruções bem definidas e não ambíguas, sendo que uma pode ser executada mecanicamente num período de tempo finito com uma quantidade finita de esforço. Um algoritmo está sempre associado a um dado objetivo, mais especificamente à solução de um problema.

Para descrever um algoritmo de forma que um computador possa compreendê-lo e executá-lo, é necessário utilizar uma linguagem de programação. Um programa é a formalização de um algoritmo numa linguagem de programação particular para que possa ser executado num computador. Sendo assim, pode-se estabelecer que a diferença entre algoritmo e programa se manifesta da seguinte forma: o primeiro é a representação da solução para um dado problema para uma pessoa, o segundo nada mais é que a representação da mesma solução voltada para um computador.

Skiena (1998, p. 9) estabelece que um algoritmo pode ser expresso através de 3 formas: (1) linguagem natural, (2) pseudocódigo ou (3) linguagem de programação executável. A linguagem natural, embora seja a linguagem com a qual o “homem” está mais familiarizado, não é recomendada quando se deseja expressar idéias algorítmicas



livres de ambigüidades; para tanto, é recomendável uma linguagem mais compacta e precisa.

Variadas formas de linguagens podem ser utilizadas para este objetivo, inclusive as próprias linguagens de programação que permitem a tradução de algoritmos para uma forma que o computador possa executar. De acordo com Shackelford (1998), o uso de linguagens de programação, apesar de permitir a execução dos algoritmos implementados, traz uma série de inconvenientes quando se tem como objetivo introduzir conceitos algorítmicos:

- As linguagens de programação são criadas com alguns objetivos específicos em mente e, dessa forma, enfatizam alguns aspectos em detrimento de outros. Não há uma linguagem que seja a “melhor”. O uso de qualquer uma resultaria num compromisso que obscurece a diferença entre propriedades importantes dos algoritmos e propriedades de certas formas de programa.
- As linguagens de programação incluem numerosos detalhes referentes a questões técnicas que introduzem complicações extras não relacionadas aos aspectos lógicos da solução do problema, e tendem a causar confusão tanto nos algoritmos como na mente de quem os implementa.
- Estudantes na fase introdutória podem ficar tão envolvidos com os mecanismos que façam seus programas “rodarem”, que perdem a compreensão da importância de um bom projeto e da “corretude” lógica. Por esta razão, é recomendável não fazer uso de uma linguagem que possa se facilmente traduzida para execução, de forma a manter a atenção do estudante focada no algoritmo ao invés de distraí-lo com minúcias de como colocar o programa para funcionar.

5.2 A Abordagem Dada ao Ensino de Programação nos Cursos de Algumas Instituições Brasileiras

Tipicamente, o ensino da programação se dá através de um encadeamento de várias disciplinas iniciando com o ensino de algoritmos, seguindo com o ensino de algumas linguagens de programação intercaladas pelo ensino de técnicas de programação e

estrutura de dados. Isto pode ser verificado na grade curricular de diversos cursos de graduação voltados para a área de informática disponível na internet.

Segundo Timóteo e Brasileiro (1998), a forma como o ensino da programação é praticada pode ser resumida da seguinte maneira:

- o curso é composto, principalmente, de aulas teóricas expositivas, com utilização de quadro negro e giz;
- as aulas práticas consistem no desenvolvimento de trabalhos específicos em laboratório;
- algumas instituições não trabalham com aulas práticas de laboratório, sendo as mesmas “compensadas” pelo livre acesso aos equipamentos, para trabalhar listas de exercícios propostas durante as aulas teóricas;
- os problemas tratados têm complexidade crescente, sendo o curso iniciado com problemas simples que vão evoluindo à medida que os conteúdos vão sendo tratados - abordagem convencionada como *bottom-up*;

Voltando a atenção ao conteúdo abordado nas disciplinas relacionadas à programação, embora o Currículo de Referência (SBC, 1999) e as Diretrizes Curriculares (MEC, 1998) sirvam de base para a elaboração das grades curriculares, muitos cursos ainda não adequaram os seus currículos às recomendações contidas nestes documentos. Resultados da pesquisa realizada por Timóteo e Brasileiro¹⁷ (1998) apontam que na prática não existe uma padronização do conteúdo abordado nestas disciplinas.

Mais especificamente, nas disciplinas que introduzem o estudante na atividade de programação, tais fatos também podem ser evidenciados. O quadro 5.1 apresenta a ementa das disciplinas em questão, sendo todas elas oferecidas no primeiro período letivo. Foram incluídas neste quadro as ementas dos cursos de algumas instituições consideradas de tradição na área de informática. Os dados foram obtidos através da internet nas páginas de divulgação dos cursos das respectivas instituições.

¹⁷ Esta pesquisa inclui uma amostragem de cursos oferecidos em 17 instituições de ensino brasileiras.



Instituição	Cursos	Disciplina	Créd.	Ementa
UFSCar	Bach. C. Comp. Eng. Comp.	Construção de Algoritmos	4	Características Básicas dos Computadores: Unidades Básicas, Instruções, Programas Armazenados, Endereçamento, Linguagens de Programação, Linguagens de Máquina, Sistemas Operacionais; Conceitos de Metodologias de Programação: Estruturada, Modular, etc; Construção de Algoritmos; Conceitos de Estratégias de Programação: Programação Independente de Linguagem, Reaproveitamento de Código, Abstração de Controle e de Dados, etc.
UNICAMP	Bach. C. Comp. Eng. Comp.	Algoritmos e Programação de Computadores	6	Fundamentos de algoritmos e sua representação em linguagens de alto nível. Estudo pormenorizado de uma ou mais linguagens. Desenvolvimento sistemático e implementação de programas. Modularidade, depuração, testes e documentação de programas.
UFPE	Bach. C. Comp. Bach. Sist. Comp. Bach. Sist. Info. Lic. C. Comp.	Introdução à Programação	6	Conceitos básicos de linguagens de programação. Conceitos básicos de programação orientada a objetos e qualidade de software. Estruturas de controle e estruturas de dados compostas. Ambiente de programação. Conceitos básicos de programação imperativa. Padrões e arquiteturas de software convencionais. Estilo de programação. Projeto de implementação
USP São Carlos	Bach. Matemática Lic. Matemática Bach. Mat. Aplicada e Comp. Científica Bach. C. Comp. Bach. em Informática	Introdução à Ciência da Computação I	6	Conceitos introdutórios de: hardware e seus componentes, sistemas operacionais, linguagens de programação e compiladores. Representação interna dos dados. Resolução de problemas e desenvolvimento de algoritmos: análise do problema, estratégias de solução, representação, e documentação. Estruturação de programas. Tipos de dados simples e avançados: vetores, matrizes, cadeias de caracteres e registros, conjuntos, arquivos e estruturas dinâmicas (ponteiros), e suas aplicações. Procedimentos, funções e passagem de parâmetros. Modularização de programas. Conceito de recursão e sua aplicação. Tipos de dados avançados. Ensino de uma linguagem estruturada (Pascal). Depuração e documentação de programas. Laboratórios de programação.
UFRGS	Bach. C. Comp. Eng. Computação	Algoritmos e Programação	6	Noção de algoritmo, dado, variável, instrução e programa; construções básicas: atribuição, leitura e escrita; estruturas de controle: sequência, seleção e iteração; tipos de dados escalares: inteiros, reais, caracteres, intervalos e enumerações; tipos estruturados básicos: vetores, matrizes, registros e strings;

Instituição	Cursos	Disciplina	Créd.	Ementa
				subprogramas: funções, procedimentos e recursão; arquivos.
IME – USP	Bach. C. Comp. Bach. Matemática Bach. Mat. Aplicada Bach. Estatística	Introdução à Computação	4	Algoritmos: caracterização, notação, estruturas básicas. Computadores: unidades básicas, instruções, programa armazenado, endereçamento, programas em linguagem de máquina. Conceitos de linguagens algorítmicas: expressões; comandos sequenciais, seletivos e repetitivos; entrada/saída; variáveis estruturadas; procedimentos. Desenvolvimento e documentação de programas. Exemplos de processamento não numérico.
PUC-RIO	Bach. Sist. Info. Eng. Computação	Técnicas de Programação I	6	O conceito de algoritmo na solução de problemas. Especificação de soluções em linguagem informal. Princípios de programação estruturada. Conceitos básicos de uma linguagem de programação: tipos de dados simples, expressões, comandos, funções. Recursividade. Vetores e matrizes. Algoritmos de ordenação. Pesquisa sequencial e binária.
UFMG	Bach. C. Comp. Matemática Computacional	Algoritmos e Estruturas de Dados I	4	Desenvolvimento de computadores e de linguagens de computação. Fases do desenvolvimento de programas. Desenvolvimento de programas em uma linguagem de alto nível: dados, comandos, ferramentas de modularização, metodologias de desenvolvimento.
UFRJ	Bach. Informática	Computação para Informática	6	Estrutura básica de um computador. Conceito de algoritmos. Blocos Fundamentais de Programação Estruturada. A Linguagem PASCAL: Estrutura de um programa, declarações e comandos principais. Diagramas de sintaxe. Representação de dados na forma de vetores, matrizes, registros e conjuntos. Procedimentos, Funções e passagem de parâmetros. Arquivos em PASCAL. Uso de ponteiros. Algoritmos numéricos e não numéricos para a solução de problemas simples.
UNESP São José do Rio Preto	Bach. C. Comp.	Programação I	4	Conceitos básicos sobre os computadores e sua programação. Construção de algoritmos usando técnicas de programação estruturada. Estruturas básicas de programação. Tipos de dados homogêneos
Abreviações: Bach.=Bacharelado / C=Ciência / Comp=Computação / Eng=Engenharia / Info=Informação / Lic=Licenciatura / Mat=Matemática / Sist=Sistemas				

Quadro 5.1 – Comparação da disciplina entre diversas instituições tradicionais



A despeito da denominação e do número de créditos que estão claramente evidenciados, outros aspectos referentes ao conteúdo abordado na disciplina contrastam entre si e com as recomendações do MEC (1998) e da SBC (1999).

Muitos cursos já abordam o ensino de programação utilizando uma linguagem de programação executável, apesar de Azeredo (2000) considerar que “... antes de travar contato com uma linguagem de programação concreta (executável)...” é necessário que o estudante tenha passado pelo “... aprendizado dos conceitos de algoritmo e fundamentos de lógica...”. São os casos, por exemplo, da UFMG (linguagem Pascal), UFRGS (Pascal), UFRJ (Pascal), IME-USP (C) e USP-São Carlos (Pascal) que fazem uso de uma linguagem de programação durante toda a extensão do curso¹⁸. Alguns deles deixam isto bem claro na própria ementa, como a USP-São Carlos e a UFRJ, que incluem o ensino da linguagem Pascal.

Partindo de um conteúdo comum que abrange os construtores básicos de uma linguagem de programação (seqüência, atribuição, seleção, iteração, modularização), a diversidade de outros conceitos abordados se faz presente de uma ementa para outra. Alguns cursos chegam a abordar arquivos, outros já chegam a algoritmos de ordenação, recursividade, estruturas dinâmicas (ponteiros) e até linguagem de máquina.

Não há um consenso geral quanto ao paradigma de programação a ser abordado inicialmente. Basicamente, duas escolas se fazem presentes: a imperativa (*procedural*) e a orientada a objetos. Existe uma tendência de se iniciar com o paradigma imperativo para depois abordar o orientado a objeto, no entanto, alguns já optam pelo paradigma orientado a objeto logo de início, como é o caso da UFPE, onde a linguagem Java faz parte do conteúdo programático.

A pesquisa realizada por Timóteo e Brasileiro (1998), aponta que grande parte dos docentes tem optado pelo paradigma imperativo, sendo a linguagem Pascal adotada como a primeira linguagem de programação ensinada aos alunos; os paradigmas funcional e lógico são abordados posteriormente ao imperativo e ao orientado a objeto.

5.3 Uma Experiência Realizada na Faculdade de Ciências Exatas e Tecnológicas da Universidade de Marília

A região de abrangência da Universidade de Marília conta com um parque industrial e comercial expressivo, com alguns destaques em nível estadual e nacional. Neste contexto, a tecnologia apresenta um papel de destaque: sistemas de processamento de dados e sistemas de informação constituem-se como recursos indispensáveis no processo administrativo e produtivo de várias empresas.

Foi neste cenário que surgiu a Faculdade de Ciências Exatas e Tecnológicas (FCET) com os cursos de Bacharelado em Ciência da Computação e Tecnologia em Processamento de Dados, implantados no início de 1988 e reconhecidos em 28/11/1991 pelas portarias nº 2250 e nº 1149 respectivamente.

No ano de 1999, com base nas recomendações da CEEInf (Comissão de Especialistas de Ensino de Computação e Informática do MEC) e da SBC (Sociedade Brasileira de Computação) estabelecendo que os cursos de graduação em informática deveriam respeitar as denominações Engenharia da Computação, Bacharelado em Ciência da Computação, Bacharelado em Sistemas de Informação e Licenciatura em Computação, a FCET, fazendo uso de sua autonomia universitária, converteu o curso de Tecnologia em Processamento de Dados para o atual Bacharelado em Sistemas de Informação, criando uma nova grade curricular e um projeto pedagógico. Juntamente com esta conversão, o curso Bacharelado em Ciência da Computação também teve sua grade reestruturada e, de integral, passou a ser oferecido nos períodos noturno e matutino. Essas reformulações passaram a vigorar a partir do ano 2000, sendo oferecidas 80 vagas para ambos os cursos, nos períodos matutino e noturno.

É justamente a partir desta data que se inicia o estudo realizado sobre o ensino da lógica de programação. O cenário que compõe a investigação conta com a participação

¹⁸ Isto pode ser verificado através da análise dos conteúdos programáticos das disciplinas em questão disponíveis nas páginas de divulgação de seus cursos (nem todos os cursos relacionados no quadro apresentam estes dados disponíveis).

dos alunos dos cursos de Ciência da Computação e Sistemas de Informação da FCET da Universidade de Marília. Esse estudo divide-se em três partes inter-relacionadas:

- procedeu-se um levantamento através de um questionário para traçar o perfil dos calouros dos cursos em questão;
- com o mesmo questionário, buscou-se realizar um estudo sobre a disciplina Construção de Algoritmos sob o ponto de vista do estudante; estudo esse complementado através de uma análise de registro que buscou obter um retrato fiel do seu desempenho na disciplina em questão, contrastando com o desempenho nas outras disciplinas oferecidas no primeiro semestre letivo dos cursos;
- realizou-se um estudo através de experimentação, onde foi avaliada a efetividade do uso da ferramenta AVA no processo de ensino-aprendizagem de lógica de programação, analisando-se aspectos como índice de aprovação/reprovação, avanço no conteúdo programático e grau de motivação dos alunos, dentre outros, apresentados pelas turmas que ingressaram em 2001 em contraste com a turma de 2000.

5.3.1 Perfil dos Alunos dos Cursos de Ciência da Computação e Sistemas de Informação na Disciplina de Construção de Algoritmos.

Com o objetivo de traçar o perfil dos calouros da FCET, um questionário foi elaborado e distribuído aos que ingressaram nos cursos de Ciência da Computação e Sistemas de Informação nos anos 2000 e 2001. A distribuição deste questionário deu-se no final do 1.o semestre de 2001.

Esta pesquisa utilizou um mecanismo de auto-seleção, onde os estudantes receberam um convite e puderam decidir quanto à sua participação. Conseqüentemente, é possível que nem todos os segmentos de dados e de opiniões estejam representados nos resultados desta investigação, o que pode vir a reduzir substancialmente o grau de generalização dos dados.



Foram consultados alunos atualmente matriculados em seis turmas¹⁹, a saber:

- Ciência da Computação (3º período letivo/matutino) – 34 alunos;
- Ciência da Computação (3º período letivo/noturno) – 78 alunos;
- Ciência da Computação (1º período letivo/matutino) – 32 alunos;
- Ciência da Computação (1º período letivo/noturno) – 80 alunos;
- Sistemas de Informação (3º período letivo/noturno) – 35 alunos;
- Sistemas de Informação (1º período letivo/noturno) – 24 alunos;

Do total de 283 alunos inicialmente matriculados, 154 (54,4%) participaram da pesquisa respondendo ao questionário (veja anexo V).

Durante a tabulação dos dados obtidos, as respostas passaram por um critério de validação e algumas delas foram ignoradas²⁰; alguns estudantes também deixaram de responder a algumas questões. Em ambos os casos, a percentagem foi calculada sobre o total de respostas válidas (os quadros com a tabulação de todas as respostas encontram-se no anexo V).

Através da tabulação das respostas, chegou-se aos seguintes dados:

- De um modo geral, as turmas são constituídas predominantemente por alunos do sexo masculino (78,8%) e solteiros (88,3%). O curso de Sistemas de Informação é que concentra a maioria das mulheres (35,3% do total de alunos do curso em questão).
- A idade média dos alunos é de 21,88 anos, sendo que 74,0% do total concentram-se na faixa entre 18 e 22 anos.
- Uma parte significativa dos alunos concluiu 1º e 2º graus em escola pública (73,4% e 58,4 respectivamente).

¹⁹ O número total de alunos expresso por turma inclui também aqueles que desistiram do curso no decorrer do período letivo.

²⁰ Algumas questões, por exemplo, as 18, 19 e 22 devem ser respondidas em duas fases, sendo a segunda fase dependente afirmativa ou negativa da primeira. Respostas de alunos matriculados no curso de Tecnologia em Processamento de Dados foram descartadas.



- O desejo de exercer a profissão não se manifestou precocemente no estudante, uma vez que 77,9% concluíram o segundo grau normal/científico, e apenas 9,8% tinham formação técnica em processamento de dados.
- A grande maioria frequenta os cursos no período noturno (73,4%), e 54,5% exercem uma atividade remunerada. No entanto, a família ainda contribui de forma significativa para o custeio do estudo (56,5%). Existe uma pequena parcela que recebe subsídio da empresa onde trabalha, bolsa de estudo da própria instituição de ensino (é o caso de alunos funcionários e alunos que desempenham papel de monitores) ou do FIES (Programa de Financiamento Estudantil patrocinado pelo governo brasileiro).
- Há um predomínio de alunos não nativos do município de Marília (apenas 37,2% são nativos). 28,8% vieram de outras cidades e residem em Marília para cumprir os estudos e viajam nos fins de semana e feriados para sua cidade natal; 34,0% residem em cidades vizinhas e viajam todo dia para assistir às aulas.
- A opção pelos cursos foi motivada pela afinidade que os estudantes afirmam ter com a informática e pela expectativa de ingressar no mercado de trabalho com maior facilidade (49,7% e 32,2% respectivamente), entre outros fatores.
- A grande maioria (70,1%) ingressou nos cursos já no seu primeiro vestibular e optou pelo Bacharelado em Ciência da Computação (74,7%).

Uma análise mais profunda das repostas revela que os calouros da FCET não provêm de famílias de nível sócio-econômico tão favorecido quanto os de outros cursos como fisioterapia, odontologia e medicina oferecidos em período integral e cuja mensalidade apresenta custo mais elevado. Isto se conclui em função da grande concentração de alunos no período noturno, embora os cursos sejam oferecidos no período matutino (aqueles que ainda não trabalham têm esperança de conseguir um emprego) e pelo fato de a grande maioria ter frequentado o ensino fundamental e médio em escolas públicas.

Depreende-se, pelo baixo número de matrículas no curso de Sistemas de Informação, que o aluno não possui um conhecimento prévio sobre este curso. Tal fato

foi constatado através de conversas informais com os calouros nas aulas iniciais, em que são discutidos os objetivos do curso e o perfil do egresso.

O fato de haver um número considerável de calouros que trabalham e outros que viajam diariamente de cidades vizinhas para estudarem a noite, faz com que estes não disponham de tempo suficiente para estudos complementares fora do horário de aula, o que vem a influir de forma negativa no seu desempenho.

5.3.2 A Disciplina Construção de Algoritmos

De acordo com o objetivo estabelecido no programa da disciplina Construção de Algoritmos (veja anexo IV), esta visa a “... fornecer técnicas de desenvolvimento da lógica de programação, criar habilidades para resolução de problemas lógicos e instituir conceitos que permitam o aprimoramento do estilo e da qualidade no processo de desenvolvimento de algoritmos...”.

Com o intuito de atingir este objetivo, a ementa da disciplina tenta adotar de forma ampla e profunda as recomendações do Currículo de Referência (SBC, 1999):

Conceitos Básicos de Lógica de Programação. Tipos de Dados Primitivos. Variáveis e Constantes. Regras para Declaração de Identificadores. Estrutura Genérica de um Algoritmo. Fluxo de Dados. Atribuição de Valores. Operadores Matemáticos, Relacionais e Lógicos. Expressões Lógicas e Aritméticas. Entrada e Saída de Dados. Teste de Mesa. Tabelas Verdade. Blocos de Execução. Estruturas de Decisão. Estruturas de Repetição. Estruturas de Dados. Modularização. Estilo e Qualidade de Programação.

Quadro 5.2 – Ementário da disciplina Construção de Algoritmos

Esta disciplina é oferecida no primeiro período letivo dos cursos de Ciência da Computação e Sistemas de Informação. Com quatro créditos, atinge uma carga horária média de sessenta e oito horas.

As práticas de ensino-aprendizagem aplicadas na disciplina podem ser divididas em três fases:

- ***Apresentação da lição - exposição de conceitos e exemplos ao aluno.*** Nesta etapa, geralmente não há uma forma intensa de interação entre professor/aluno. Os meios utilizados para transmissão baseiam-se em livro-texto e em aula expositiva. O ensino é dirigido e o aluno é passivo.
- ***Realização de exercícios.*** Nesta etapa, o aluno exercita ou valida o que lhe foi exposto e aprendido. São aplicados exercícios tanto de forma dirigida, em que o aluno é mais passivo (por exemplo: exercícios de múltipla escolha), como de forma não dirigida, em que o aluno é ativo, através de exercícios de criação (construção de casos, resolução de problemas).
- ***Avaliação.*** Nesta etapa, o professor avalia o grau de assimilação dos conceitos pelo aluno e sua capacidade de aplicá-los em diferentes circunstâncias. Essa avaliação costuma ser quantitativa (apenas marcando os exercícios como corretos ou não) e qualitativa (com correção/explicação dos erros cometidos).

5.3.3 A Deficiência Apresentada na Disciplina Construção de Algoritmos

A disciplina **Construção Algoritmos** é encarada como a mais importante das que são oferecidas no primeiro período letivo, tanto por professores como pelos próprios alunos, devido ao fato de ser pré-requisito para todas as disciplinas subseqüentes que envolvem programação. O mau desempenho nesta disciplina vem a acarretar sérias conseqüências, podendo atrasar em um ano o prazo de conclusão do curso em caso de reprovação e desempenho insatisfatório nas disciplinas subseqüentes, caso seja aprovado com deficiências.

Com o objetivo de obter um retrato mais fiel da real situação dos alunos diante desta disciplina, foi feito um estudo através de análise de registros e aplicação de questionário.

Inicialmente, foi avaliado o desempenho dos alunos matriculados no primeiro período letivo de 2000 e 2001 nos cursos de Ciência da Computação e Sistemas de

Informação, onde foram estabelecidos através de análise de registros os índices de aprovação, reprovação e desistência, e o valor médio da nota final obtida.

As notas destes alunos foram obtidas através da ata de registro de avaliações junto ao sistema de controle acadêmico. O seu lançamento foi feito agrupando-os por disciplina, a saber:

- Cálculo Diferencial e Integral
- Comunicação e Expressão
- Construção de Algoritmos
- Geometria Analítica e Álgebra Linear
- Introdução à Ciência da Computação
- Matemática Discreta

Com exceção das disciplinas **Cálculo Diferencial e Integral** – oferecida somente ao curso de Ciência da Computação – e **Geometria Analítica e Álgebra Linear** – oferecida somente ao curso de Sistemas de Informação – as demais fazem parte da grade curricular de ambos os cursos.

Foi considerado desistente o aluno que não esgotou as possibilidades de avaliação ou teve frequência insuficiente (abaixo de 75% do número total de aulas). Também foram incluídos nesta classe aqueles que trancaram ou cancelaram a matrícula.

Os índices de aprovação e reprovação foram calculados com base no critério de avaliação conforme demonstrado pelas fórmulas a seguir:

- $MF_1 = (MB_1 + MB_2) / 2$
- $MB_i = AP * 0.4 + PR * 0.6$ para $i=1,2$
- $MF_2 = (MF_1 + EX) / 2$

onde MB_1 e MB_2 correspondem respectivamente às notas dos 1.o e 2.o bimestres; AP corresponde à nota da avaliação parcial (0 a 10); PR corresponde à nota da prova regimental (0 a 10); MF_1 corresponde a média final dos dois bimestres; EX corresponde à nota obtida no exame (0 a 10) e MF_2 corresponde à média final, considerando a nota obtida no exame. O quadro 5.3 estabelece a situação do aluno de acordo com o desempenho obtido nas notas.

MF ₁ ≥ 7.0 → aprovado	
MF ₁ < 4.0 → reprovado	
4.0 ≤ MF ₁ < 7.0 → direito a exame	
Desempenho no Exame	MF ₂ ≥ 5.0 → aprovado
	MF ₂ < 5.0 → reprovado

Quadro 5.3 – Critério de avaliação

Com base nesses critérios, as notas de todos os alunos referentes ao primeiro período letivo das turmas de 2000 e 2001 foram lançadas, as médias calculadas e foram obtidos os seguintes gráficos:

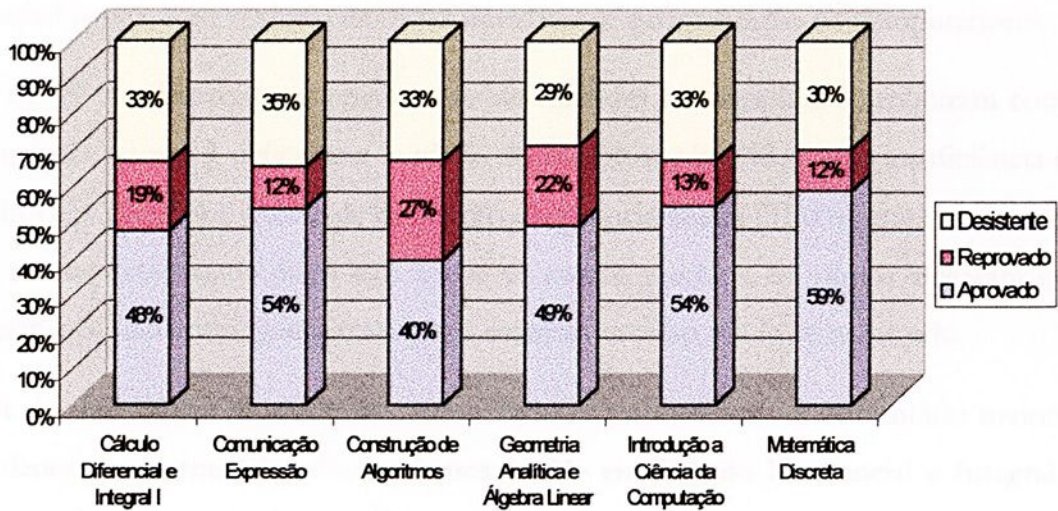


Gráfico 5.1 - Índices de aprovação, reprovação e desistência

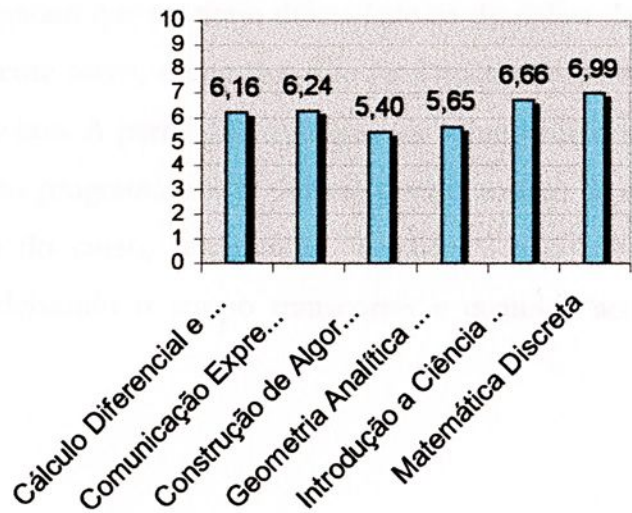


Gráfico 5.2 – Valor médio das notas nas disciplinas

Os gráficos 5.1 e 5.2 demonstram que a disciplina Construção de Algoritmos é responsável pelo maior índice de reprovação e um dos maiores de desistência, assim como pela menor nota média.

Com o intuito de estudar as causas desse baixo desempenho, no mesmo questionário utilizado para traçar o perfil dos calouros foram incluídas questões para coletar a sua opinião sobre o ensino da disciplina Construção de Algoritmos (veja anexo V, questões numeradas de 18 a 23). Através da tabulação das respostas (veja também anexo V), chegou-se aos seguintes dados:

- A grande maioria está ciente da importância desta disciplina (80,5%) como sendo primordial para inserir o aluno na difícil atividade de programação de computadores.
- Cerca de 65 % afirmaram ter encontrado dificuldades na disciplina e atribuíram como os principais fatores à deficiência herdada do ensino médio (18,0%), a insuficiência da carga horária (30,0%) e a falta de dedicação do próprio aluno (31,0%), mesmo estando ciente da importância da disciplina. Entre os outros motivos, os alunos alegaram que simplesmente não conseguiam entender a matéria, mesmo tendo se esforçado.
- Apesar da dificuldade na disciplina, cerca de 43 % afirmaram ter enfrentado maiores dificuldades em outras disciplinas. Destes 45,2% em Cálculo Diferencial e Integral e 24,2% em Geometria Analítica e Álgebra Linear.

Em conversas informais com alunos que apresentaram um desempenho insatisfatório, alguns alegaram que sentiram dificuldade na disciplina devido ao fato de ser um assunto completamente novo, e que por este fato muitos levaram um tempo maior para assimilar seus conceitos. A partir daí, conseguiram acompanhar de forma satisfatória o desenrolar do conteúdo programático. Já outros, devido ao fato de não terem encarado com seriedade o início do curso, sentiram-se “perdidos”, mas não atentaram para a gravidade da situação, deixando o tempo transcorrer e quando “acordaram” era tarde demais.

5.3.4 Análise Comparativa entre as Turmas de 2000 e 2001

Este estudo visa a avaliar a efetividade do uso da ferramenta AVA no ensino de lógica de programação. Para tanto, a ferramenta foi testada empiricamente nas turmas de **Ciência da Computação (CC)** e **Sistemas de Informação (SI)** que iniciaram em 2001.

A avaliação da ferramenta foi feita através da análise de desempenho, confrontando os índices de aprovação e reprovação. Também foi averiguado o grau de motivação dos alunos, a progressão do conteúdo programático e, a abrangência e a profundidade com que este foi abordado.

5.3.4.1 A Turma de 2000

No ano de 2000, o total de 220 alunos inicialmente matriculados foi distribuído da seguinte forma:

- CC (noturno A) – 50 alunos
- CC (noturno B) – 57 alunos
- SI (noturno) – 49 alunos
- CC (matutino) – 64 alunos

Através de uma idéia inovadora na faculdade, três professores (sempre os mesmos) foram designados para ministrar a disciplina de **Construção de Algoritmos (CA)** em todas as turmas, com o intuito de trabalhar com exercícios práticos durante 70% da carga horária total.

A prática de aula pode ser resumida como apresentação do conceito através de aulas expositivas utilizando o quadro negro, seguida de alguns exemplos e a realização de vários exercícios fazendo uso do tradicional “teste de mesa”.

O livro-texto adotado como material didático obrigatório foi o de Forbellone e Eberspacher (1999) – *Lógica de Programação – A Construção de Algoritmos e Estruturas de Dados*.

Para a exposição da aula, houve revezamento entre os três professores. Mas na prática dos exercícios todos atuavam na mesma sala de forma ativa. Foi possível fazer um

acompanhamento individual de cada aluno. O aluno implementava o algoritmo escrevendo o pseudocódigo no caderno e através do “teste de mesa” averiguava o seu funcionamento. Durante esta fase, notou-se que poucos realizavam o teste. Frequentemente, o aluno recorria ao professor para verificar se a sua implementação estava correta.

5.3.4.2 A Turma de 2001

Já no ano de 2001 a faculdade teve uma redução substancial no número de alunos matriculados no primeiro período letivo. Do total de 148 alunos, a distribuição procedeu-se da seguinte forma:

- CC (noturno) – 80 alunos
- SI (noturno) – 24 alunos
- CC (matutino) – 32 alunos

Para a turma de CC do período noturno foram designados dois professores e nos demais apenas um.

Houve uma mudança radical na prática de aula em comparação com a turma do ano anterior. A exposição da aula passou a ser feita com a utilização de recursos de videoprojeção (quando este recurso encontrava disponível) aliados à ferramenta AVA. Esta nova abordagem mostrou-se eficiente, principalmente na demonstração de exemplos. A realização dos exercícios passou a ser feita em laboratório, com o aluno utilizando interativamente a ferramenta supracitada na resolução de problemas, seguindo o paradigma construcionista.

O material didático adotado para esta turma foi baseado na pseudolinguagem implementada na ferramenta AVA e encontra-se no anexo VI.

5.3.4.3 A Comparação

Para efeito de avaliar o desempenho dos alunos, foi utilizada a mesma tabulação das notas mencionada na seção 5.3.3, separando-as pelo ano de seu ingresso.



Analisando isoladamente o desempenho dos alunos das duas turmas na disciplina CA poder-se-ia concluir que o uso da ferramenta não surtiu efeito. Os índices de aprovação apontam que não houve uma mudança significativa, pois estes permaneceram praticamente inalterados, com 59% e 61% respectivamente para as turmas de 2000 e 2001. Considerando-se o valor médio das notas pôde-se notar que houve até um declínio na turma de 2001 (5,37 contra 5,42 da turma de 2000). No entanto, se esta análise envolver as outras disciplinas é constatado que o desempenho relativo foi significativamente superior. Um indício claro que leva a esta conclusão está no desempenho geral apresentado pela turma de 2001, que foi substancialmente inferior ao da turma de 2000, subtendendo-se que a primeira é mais “fraca”. Isto pode ser constatado no confronto dos índices de aprovação das duas turmas apresentado no gráfico 5.3. Este índice foi calculado sobre o total entre aprovados e reprovados (os desistentes foram excluídos).

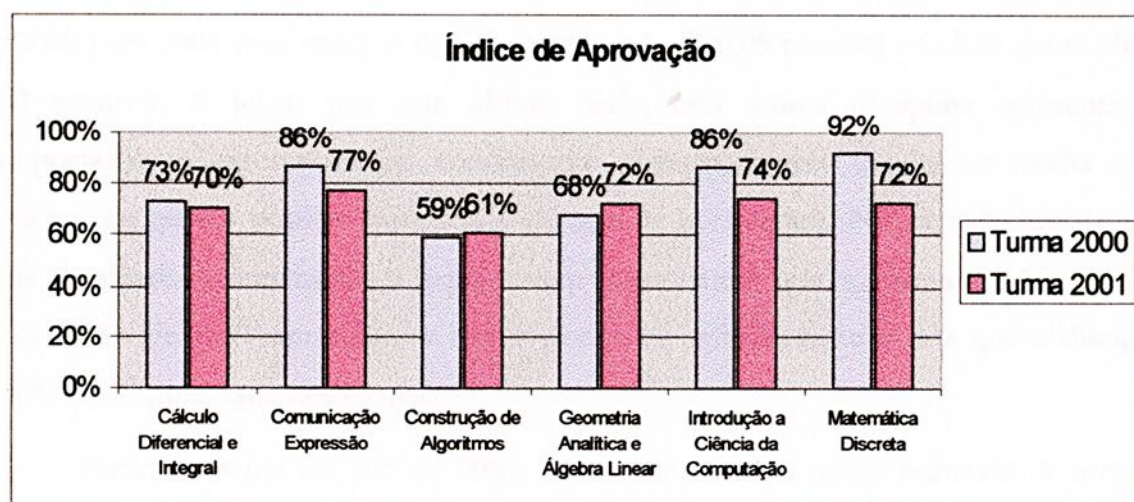


Gráfico 5.3 – Comparação entre as turmas 2000 e 2001

Pode ser verificado através do gráfico 5.3 que houve um pequeno aumento na taxa de aprovação em CA e em **Geometria Analítica de Álgebra Linear** (GAAL), enquanto nas demais disciplinas estas taxas caíram substancialmente, ou seja, a melhora no desempenho relativo foi significativa. Isto também é constatado no valor médio da nota (veja o gráfico 5.4).

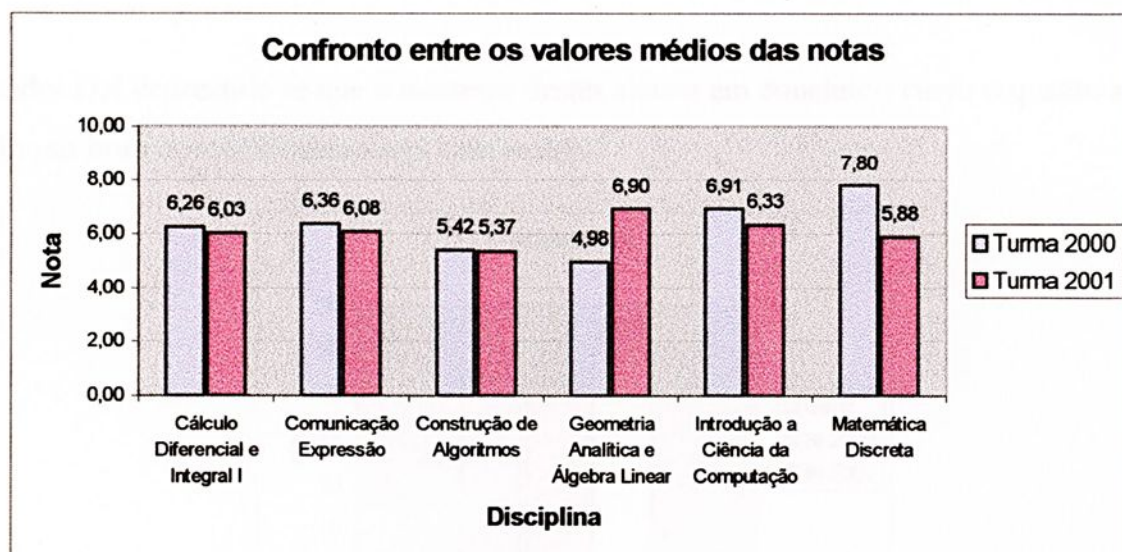


Gráfico 5.4 – Comparação entre as médias das notas da turma de 2000 e 2001

De uma forma geral, houve uma queda expressiva no valor médio das notas finais de 2000 para 2001 com exceção de CA (com queda de 0,05 pontos) e GAAL (com alta de 1,92 pontos). A julgar por este último dado, esta última disciplina apresenta um comportamento excessivamente contrastante com as demais (a alta na média e um aumento de quatro pontos percentuais na taxa de aprovação). No entanto, uma análise mais detalhada (apresentada a seguir) vem demonstrar que a turma de Sistemas de Informação de 2001 apresenta-se como ponto de referência, uma vez que a disciplina GAAL só é ministrada neste curso.

Particularmente no ano de 2001, houve um número muito reduzido de inscritos para o curso de SI, que resultou numa turma com poucos alunos (24 alunos versus 49 matriculados no primeiro período de 2000). A redução neste número permitiu aos professores oferecer uma aula de maior qualidade, que conseqüentemente resultou numa melhora de desempenho destes alunos; outros fatores que contribuíram para que a turma de SI de 2001 destoasse da média geral podem ser verificados nos gráficos 5.5, 5.6 e 5.6 (convém salientar que os dados contidos neste gráfico foram obtidos considerando-se uma amostra do total de alunos).

Os alunos da turma de SI de 2001 apresentam a maior idade média em comparação com a média geral e a média da turma de SI de 2000. Também apresentam um número maior de alunos que trabalham e conseqüentemente arcam com o custeio do próprio

estudo. Daí depreende-se que o interesse destes alunos em concluir o curso o quanto antes com um bom aproveitamento seja bem maior.

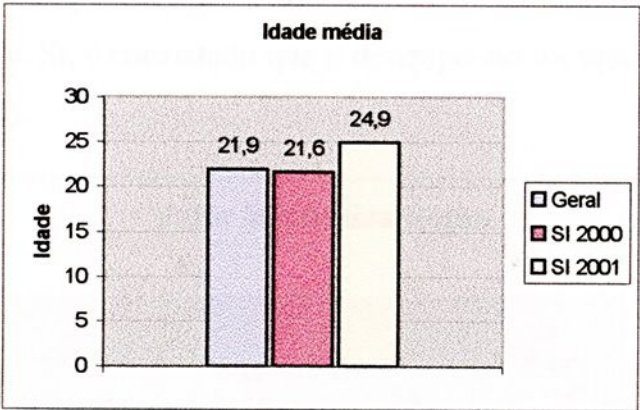


Gráfico 5.5 – Comparação entre as idades médias

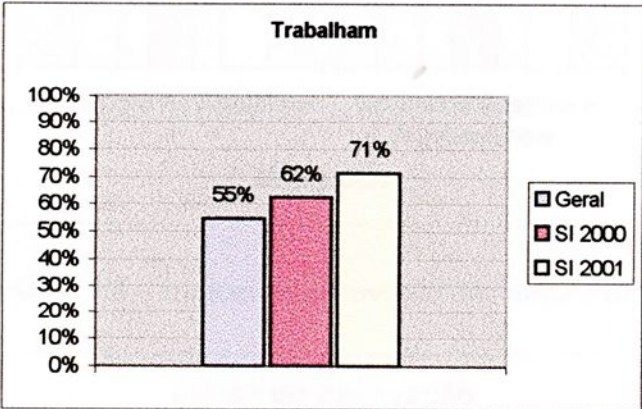


Gráfico 5.6 – Porcentagem de alunos que trabalham

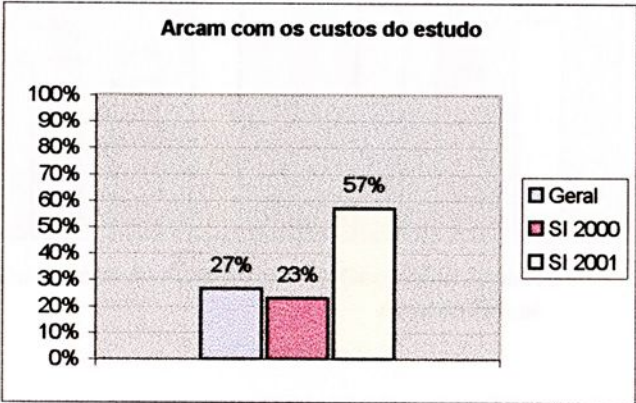


Gráfico 5.7 – Porcentagem de alunos que arcam com o custeio do estudo

Dentro deste contexto, analisando isolamente somente as turmas de SI de 2000 e 2001, é de se esperar que o desempenho da turma de 2001 seja superior. Foi o que aconteceu na disciplina GAAL no quadro geral (voltando a salientar que esta disciplina só é oferecida ao curso em questão). Da mesma forma, se for considerada a disciplina CA somente nas turmas de SI, é constatado que o desempenho foi significativamente superior (veja gráficos 5.8 e 5.9).

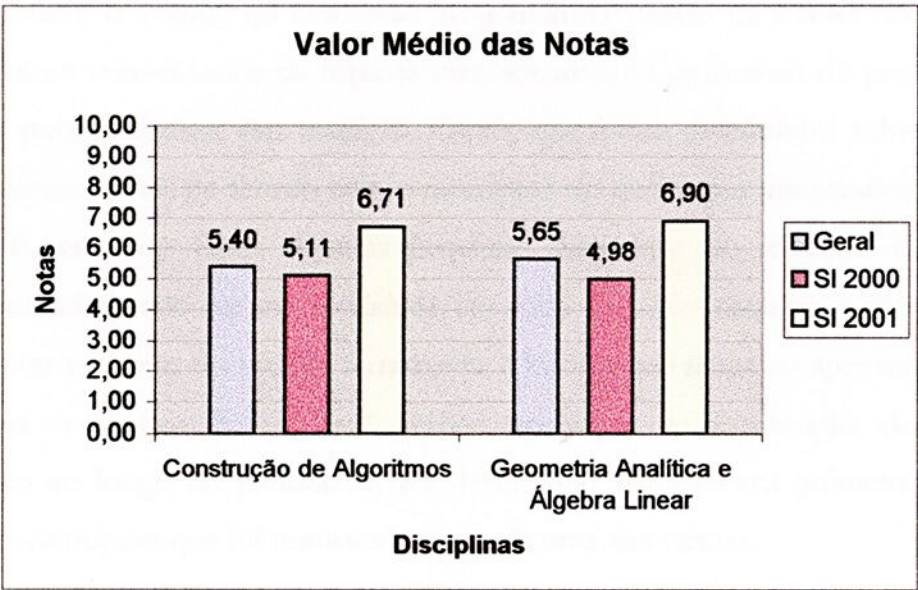


Gráfico 5.8 – Índices de aprovação das turmas de SI

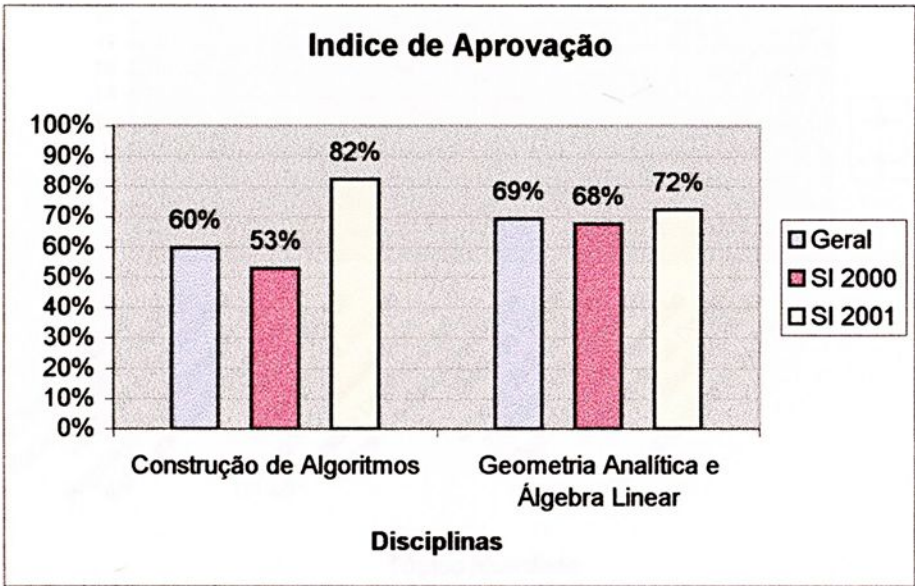


Gráfico 5.9 – Valor médio das notas das turmas de SI

Levando-se em conta todos estes fatos, no cômputo geral de toda a estatística realizada pode-se afirmar categoricamente que o desempenho relativo alcançado pela turma de 2001 na disciplina CA supera a de qualquer outra, subtendendo-se que o uso da ferramenta AVA trouxe um ganho satisfatório ao processo de ensino-aprendizagem, no que diz respeito ao desempenho do aluno.

Para avaliar mais profundamente a efetividade da ferramenta, foi realizado também um estudo sobre o avanço do conteúdo programático através da análise do registro de matérias. Foram considerados os tópicos mencionados no ementário do programa (veja quadro 5.2) para se realizar esta medição. Os tópicos foram distribuídos sobre o número total de semanas letivas, de acordo com o momento em que foram ministrados nas turmas (total de 18 semanas). Com algumas pequenas diferenças no intervalo de tempo o conteúdo foi ministrado na sua totalidade em cada ano. No entanto, a turma de 2001 conseguiu tirar proveito do uso da ferramenta AVA no que tange ao aproveitamento do tempo (veja o gráfico 5.10). Este gráfico apresenta a distribuição do conteúdo programático ao longo do período letivo, destacando o momento (número da semana dentro do período) em que foi ministrado em cada uma das turmas.

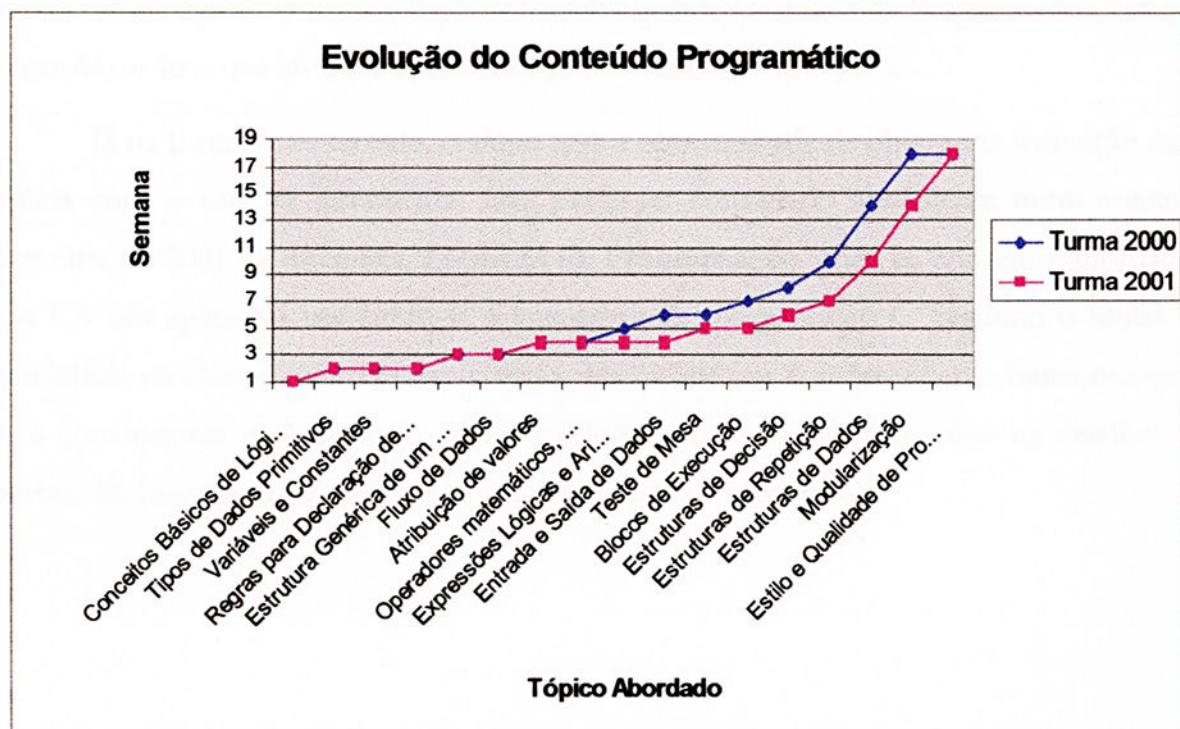


Gráfico 5.10 – Evolução do conteúdo programático

Até a quarta semana o conteúdo evoluiu igualmente, considerando-se que os tópicos abordados até então foram praticados sem a utilização da ferramenta AVA (na turma de 2001). A partir do momento que a ferramenta pôde ser utilizada tanto na exposição da aula como na prática de exercícios por parte do aluno, além de o conteúdo poder ser abordado de forma mais rápida, o aluno teve condições de testar sua implementação interativamente, visualizando o seu funcionamento. Este fato trouxe uma motivação a mais ao aluno, que passou a praticar os exercícios fora dos horários de aula também; muitos deles solicitaram uma cópia da ferramenta para que pudessem instalar em seus computadores na sua residência e no local de trabalho. Dessa forma, o conteúdo pôde ser cumprido com folgas no prazo, restando um tempo maior para a prática de exercícios.

Em oposição, a turma de 2000 apresentou algumas deficiências que podem ser constatadas através do gráfico 5.10. Os tópicos **Modularização** e **Estilo e Qualidade de Programação** foram abordados somente na última semana, inviabilizando a realização de exercícios de fixação. Aspectos importantes referentes a estes tópicos como **função** e **parâmetros** só foram mencionados para constar do registro de matéria. Tal fato resultou numa sobrecarga de conteúdo da matéria subsequente (Técnicas de Programação), em que o professor teve que abordar com maior profundidade estes tópicos.

Já na turma mais recente, o aluno teve a oportunidade de obter uma formação mais sólida com o uso da ferramenta. Isto pode ser constatado atualmente neste segundo semestre de 2001 na disciplina **Técnicas de Programação**, onde os conceitos abordados em CA são aplicados utilizando-se a linguagem de programação C. Segundo o titular da disciplina, os alunos não têm encontrado dificuldade em compreender o funcionamento dos construtores da linguagem, visto que logo após à explanação sobre os detalhes de sintaxe da linguagem eles partem para a resolução dos exercícios.

Considerações Finais

A incorporação da tecnologia de computadores no âmbito educacional está dando oportunidade às escolas de criarem uma nova forma de trabalhar os conteúdos programáticos, propiciando ao educando eficiência na construção do conhecimento, convertendo a aula num espaço real de interação, de troca de resultados, adaptando os dados à realidade do educando.

No entanto, como acontece com todas as ferramentas alternativas, a proposta do uso de computadores para auxiliar no processo de ensino-aprendizagem, de uma forma geral, requer certos cuidados para que seus objetivos sejam realmente alcançados e para que todos os esforços não produzam consequências negativas.

No presente trabalho, após constatada a deficiência no ensino de programação no tocante à forma de exposição da aula baseada em quadro-negro e giz e, a prática do aluno baseada em “teste de mesa”, foi implementado um sistema de visualização de software visando a dar uma abordagem alternativa à prática de ensino do assunto em questão.

Valendo-se das características do computador (alta capacidade de armazenamento e processamento de dados aliado aos recursos de multimídia) a utilização de um sistema computadorizado proporcionou o surgimento de um ambiente diferente dos modelos utilizados em aulas tradicionais, tornando-se altamente estimulante ao aluno, permitindo-lhe interagir diretamente com suas implementações.

Para aproveitar as valências propiciadas pelo uso do computador aliado ao software (ferramenta AVA), houve uma necessidade premente de repensar as práticas de ensino, pois o professor deixa de ser o centro das atenções, o detentor e o fornecedor do conhecimento. A partir deste momento, ele transforma-se numa espécie de

condutor/facilitador, estimulando seus alunos a navegarem pelo conhecimento e a fazerem suas próprias descobertas.

A Experiência com o Uso da Ferramenta AVA

A utilização da ferramenta na disciplina Construção de Algoritmos deu um novo alento às práticas de ensino-aprendizagem, oferecendo ao aluno oportunidade de experimentar na prática os algoritmos desenvolvidos. Isso permitiu que a abordagem dada à disciplina continuasse sendo feita através do uso de uma pseudo-linguagem em detrimento de uma linguagem de programação convencional (como Pascal e C).

Sem a menor sombra de dúvida, o maior benefício obtido com o uso da ferramenta AVA foi a motivação dos alunos em poder abordar de uma forma diferente um assunto que até então era tratado somente em sala de aula, já que uma das grandes reivindicações dos calouros era o aumento do número de aulas em laboratório.

O maior contraste constatado durante a realização da experiência com os alunos das turmas de 2000 e 2001 pôde ser notado durante as aulas de resolução de exercícios. O ambiente construcionista proporcionado pelo AVA permitiu ao aluno descrever, depurar e refletir sobre a solução do problema sendo implementado, tornando o processo de aprendizagem mais significativo e, conseqüentemente, propiciando a construção de seu conhecimento.

Com a utilização da ferramenta durante todo um período letivo, alguns pontos-chave puderam ser identificados:

- pronto feedback: durante o processo de desenvolvimento e execução de um algoritmo, o sistema pode acusar imediatamente uma ação executada pelo usuário que seja considerada errada, sem o ônus do uso de caneta e papel (tradicional “teste de mesa”) que consome tempo, exige freqüentes transcrições, não permitem fáceis manipulações ou acréscimos textuais;
- controle do ritmo de aprendizagem: o sistema respeita o ritmo de aprendizagem do aluno, pois este passa a controlar a velocidade com que a aprendizagem deve se dar;

- desenvolvimento do algoritmo por tentativa e erro: uma parcela dos alunos passou a utilizar o método da “tentativa e erro” justamente pela facilidade em testá-los, fugindo da responsabilidade de elaborar uma solução para depois testá-la;
- conexão entre a representação mental e a visualização: alguns alunos sentiram dificuldade inicial na utilização da ferramenta pelo fato de não conseguirem abstrair a visualização da execução do algoritmo para o seu modelo mental.

Apesar de alguns percalços, a ferramenta AVA mostrou a sua efetividade conforme pôde ser constatado na experiência realizada.

Reestruturação do Ambiente de Ensino-Aprendizagem

Muitos educadores e especialistas em tecnologia acreditam no imenso potencial do computador como ferramenta de ensino. No entanto, para perceber este potencial é preciso repensar “o que ensinar”, “como ensinar” e, mais do que tudo, “como aprender” para adaptar as práticas tradicionais de ensino-aprendizagem ao mundo da tecnologia. Existe uma diversidade de recursos tecnológicos da qual o ensino pode valer-se, e um número grande de pesquisas está sendo realizado, visando sempre à qualidade no ensino.

O professor deve possuir o discernimento sobre o potencial educacional dos computadores e ser capaz de alternar adequadamente atividades tradicionais de ensino-aprendizagem e atividades que usam o computador. A atividade de uso do computador pode ser feita tanto para continuar transmitindo a informação para o aluno e, portanto, para reforçar o processo instrucionista, quanto para criar condições para que ele possa construir seu conhecimento por meio da criação de ambientes de aprendizagem que incorporem o uso do computador (VALENTE, 1993a).

Sob este cenário, as práticas atuais devem sofrer profundas alterações, tanto por parte do aluno como pelo professor. Os alunos terão muito mais responsabilidade sobre o gerenciamento de suas tarefas do que no modelo tradicional e o papel do professor passa a ser o de orientador ou facilitador. A tecnologia instrucional tradicional é baseada num relacionamento linear e hierárquico, enquanto que a tecnologia educacional emergente revoluciona o processo ensino-aprendizagem (veja quadro seguir).

	<i>Tecnologia Instrucional Tradicional</i>	<i>Tecnologia Instrucional Emergente</i>
<i>Papel do professor</i>	Especialista	Facilitador
<i>Papel do aluno</i>	Receptor passivo	Colaborador ativo
<i>Ênfase instrucional</i>	Fatos e Aprendizagem dirigida	Pensamento crítico
<i>Avaliação da aprendizagem</i>	Retenção	Assimilação e interpretação de fatos
<i>Método de ensino</i>	Exercício e Prática	Interatividade e colaboração
<i>Acesso ao conhecimento</i>	Acesso limitado ao conhecimento e informação	Acesso ilimitado ao conhecimento e à informação via tecnologia

Quadro – Comparação entre a tecnologia instrucional emergente e tradicional
(HOSSAIN apud CAMPOS et al., 1998)

Apesar dessa iminente revolução, ao contrário do que pensam muitos educadores, o computador está longe de ser um substituto do professor na sala de aula; ele deve ser considerado uma ferramenta de auxílio, complementação e aperfeiçoamento, que possibilite melhorias na qualidade de ensino.

O professor será mais importante do que nunca, pois ele precisa apropriar-se dessa tecnologia e introduzi-la tanto na sala de aula como no seu dia-a-dia, da mesma forma que introduziu o primeiro livro numa escola e teve que começar a lidar de modo diferente com o conhecimento, sem deixar as outras tecnologias de comunicação de lado.

Sendo assim, cabe ao educador não ficar alheio à utilização desses recursos e reformular a sua maneira de atuar em sala de aula. A busca por novas alternativas que premiem o aluno com a melhora na qualidade de ensino deve ser sempre incessante para o educador competente.



Bibliografia

- AHO, A. V., ULLMAN, J. D., SETHI, R. (1995). *Compiladores – Princípios, Técnicas e Ferramentas*. 1.a Edição, Editora LTC, 344 p.
- AZEREDO, P. A de (2000). *Proposta de Plano Pedagógico para a Matéria de Programação*. in Anais do II Curso: Qualidade de Cursos de Graduação da Área de Computação e Informática – VIII Workshop sobre Educação em Computação, Curitiba, pp.1-13
- BAECKER, R. (1998). *Sorting Out Sorting: A Case Study of Software Visualization*. in Stasko, J., Domingue, J., Brown, M.H., Price, B.A. (org.) **Software Visualization: Programming as a Multimedia Experience**. Massachusetts Institute Technology Press, pp.369-381
- BAECKER, R., PRICE, B. A. (1998). *The Early History of Software Visualization*. in Stasko, J., Domingue, J., Brown, M.H., Price, B.A. (org.) **Software Visualization: Programming as a Multimedia Experience**. Massachusetts Institute Technology Press, pp.81-101
- BAZIK, J., TAMASSIA, R., REISS, S.P., DAM, A. (1998). *Software Visualization in Teaching at Brown University*. in Stasko, J., Domingue, J., Brown, M.H., Price, B.A. (org.) **Software Visualization: Programming as a Multimedia Experience**. Massachusetts Institute Technology Press, pp.384-398

- BIANCHETTI, Lucídio. (1997) *Dilemas do Professor Frente ao Avanço da Informática na Escola* in Boletim Técnico do Senac, vol. 23, nº 2, maio/agosto-1997, p. 9 (versão eletrônica disponível em <http://www.senac.br/boletim/boltec21.htm>).
- BIRCH, M.R. (1990). *An Emulator for the E-Machine*. (Dissertação de Mestrado em Ciência da Computação) - Montana State University (versão eletrônica disponível em <http://esus.cs.montana.edu/~dynamalab/publications/theses/theses.html>)
- BIRCH, M.R., BORONI, C.R., Goosey, F.W., Patton, S.D., Poole, D.K., Pratt, C.M., Ross, R.J. (1995). *DYNALAB: A Dynamic Computer Science Laboratory Infrastructure Featuring Program Animation*. in Proceeding of The 26^o SIGSE – Technical Symposium on Computer Science Education, pp. 29-33
- BORONI, C.M.; ENEBOE, T.J.; GOOSEY, F.W.; ROSS, J.A.; ROSS, R.J. (1996). *Dancing With Dynalab*. in Proceedings of the 27^o. SIGCSE Technical Symposium on Computer Science Education, pp.135-139
- BROWN, M. H.(1991). *Zeus: A System for Algorithm Animation and Multi-View Editing*. in Proceedings of the IEEE Workshop on Visual Languages, IEEE Computer Society Press, Kobe, Japan, pp. 4-9 (versão eletrônica disponível em <http://citeseer.nj.nec.com/39518.html>)
- BYRNE, M. D., CATRAMBONE, R., STASKO, J. T. (1996). *Do Algorithm Animations Aid Learning?* Graphics, Visualization, and Usability Center, Georgia Institute of Technology, Atlanta, GA, Technical Report GIT-GVU-96-18. (versão eletrônica disponível em <http://www.gvu.gatech.edu/gvu/softviz/pubs/pubs.html>)
- CAMPOS, Fernanda C. A., ROCHA, Ana Regina C. da, CAMPOS, Gilda H. B. de (1998) *Design Instrucional e Construtivismo: Em Busca de Modelos para o Desenvolvimento de Software* in anais do RIBBIE98- IV Congresso Ibero-Americano de Informática na Educação, Brasília – Brasil – 20 a 23 de Outubro de 1998 (versão eletrônica disponível em <http://www.c5.cl/ieinvestiga/actas/ribie98/250M.html>)
- CRESCENZI, P., DEMETRESCU, C., FINOCCHI, I., PETRESCHI, R. (2000). *Reversible Execution and Visualization of Programs with Leonardo*. in Journal of Visual

Languages and Computing, 11(2), pp. 125-150, Academic Press (versão eletrônica disponível em

<http://www.dis.uniroma1.it/~demetres/Leonardo/Documentation.html>)

CRESPO, S., EVARISTO, J. (2000). *Aprendendo a Programar: Programando numa Linguagem Algorítmica Executável (ILA)*. Book Express

DEFANTI, Thomas A., Brown, Manine D., McCornick, Bruce H. (1999) *Visualization – Expanding Scientific and Engineering Research Opportunities* in Card, Stuart K., MacKinlay, Jock D., Shneiderman, Ben Readings in *Information Visualization: Using Vision to Think*, Morgan Kaufmann (Academic Press), 1999, pp. 39-56

ELLERSHAW, S., OUDSHOORN, M. (1994). *Program Visualization – The State of the Art*. Department of Computer Science. University of Adelaide, Technical Report 94-19 (versão eletrônica disponível em <http://citeseer.nj.nec.com/ellershaw94program.html>)

ESMIN, A. A. A.. (1998). *Portugol/Plus: Uma Ferramenta de Apoio ao Ensino de Lógica de Programação Baseado no Portugol*. in Anais do IV Congresso RIBIE, Brasília (versão eletrônica disponível em <http://www.nied.unicamp.br/ribie/trabalhos/118.pdf>)

FARRER, H., BACKER, C.G., FARIA, E.C. (1999). *Programação Estruturada de Computadores: Algoritmos Estruturados*. 3.a Edição, Rio de Janeiro, Editora LTC, 260p.

FERNANDES, A. L. B.; Botini, J. (1998) *Construção de Algoritmos*. Rio de Janeiro, Editora Senac Nacional, 112 p.

FERREIRA, A. B. de H. (1999). *Novo Aurélio – Século XXI: O Dicionário da Língua Portuguesa*. 3.a Edição, Rio de Janeiro, Nova Fronteira

FORBELLONE, A. L. V., EBERSPACHER, H. F. (1999) *Lógica de Programação – A Construção de Algoritmos e Estruturas de Dados*. 2ª Edição, Makron Books, 196 p.

- FORD, L. (1993). *Interactive Learning and Researching with Visualization*. Research Report 274, University of Exeter (versão eletrônica disponível em <http://citeseer.nj.nec.com/ford93interactive.html>)
- FREIRE, Fernanda M. P., PRADO, Maria Elisabette B. (1996) *Professores Construcionistas: A Formação em Serviço* in anais do RIBBIE96 – III Congresso Ibero-Americano de Informática na Educação, Barranquilha, Colômbia (versão eletrônica disponível em http://www.niee.ufrgs.br/ribie98/CONG_1996/CONGRESSO_HTML/64/FO RMSERV.HTML)
- GARCIA, I.C., REZENDE, P.J. de, CALHEIROS, F.C. (1997). *Astral: Um Ambiente para Ensino de Estruturas de Dados através de Animações de Algoritmos*. in Revista Brasileira de Informática na Educação No. 1 – Sociedade Brasileira de Computação. (versão eletrônica disponível em <http://www.inf.ufsc.br/sbc-ie/revista/nr1/garcia.htm>)
- GOOSEY, F. W. (1993). *A MiniPASCAL Compiler for the E-Machine*. (Dissertação de Mestrado em Ciência da Computação) - Montana State University (versão eletrônica disponível em <http://esus.cs.montana.edu/~dynamlab/publications/theses/theses.html>)
- GOUVÊIA, Sylvia Figueiredo (1999) *Os Caminhos do Professor na Era da Tecnologia* in Revista de Educação e Informática Ano 9 - número 13 - abril 1999 (versão eletrônica disponível em <http://www.educacao.sp.gov.br/publicacoes/acesso/acs13p03.htm>)
- HAREL, D. (1992). *Algorithmics, The Spirit of Computing*. 2.a Edição, Addison Wesley, 440 p.
- KEHOE, C., STASKO, J.T. (1996). *Using Animations to Learn about Algorithms: An Ethnographic Case Study*. Graphics, Visualization, and Usability Center, Georgia Institute of Technology, Atlanta, GA, Technical Report GIT-GVU-96-20. (versão eletrônica disponível em <http://www.gvu.gatech.edu/gvu/softviz/pubs/pubs.html>)

- KNUTH, D. R. (1997) *The Art of Computer Programming. Fundamental Algorithms*. Vol. 1, 3.a Edição, Addison Wesley, 650p.
- LARSEN, Steen (1999) *O Computador é um Instrumento de Laboratório* in Revista de Educação e Informática Ano 9 - número 13 - abril 1999 - tradução Dimitri Domatewicz (versão eletrônica disponível em <http://www.educacao.sp.gov.br/publicacoes/acesso/acs13p05.htm>)
- LAWRENCE, A., BADRE, A., STASKO, J. T. (1994). *Empirically Evaluating the Use of Animations to Teach Algorithms*. in Proceedings of the 1994 IEEE Symposium on Visual Languages, St. Louis, MO, pp.48-54 (versão eletrônica disponível em <http://www.gvu.gatech.edu/gvu/softviz/publs/publs.html>)
- LEITE, Aury de Sá; SILVA, César Alberto (2001) *A Aquisição de Perícia Cognitiva Apoiada por Computadores*. in Anais do I Congresso Brasileiro de Computação – CBComp2001, Universidade do Vale do Itajaí, Itajaí
- MACHADO, Nilson (1995) *Epistemologia e Didática*, Editora Cortez, 320p.
- MELEIRO, Alessandra (1997) *Novos Modos de Representação para os Modelos Teóricos para a Estrutura da Matéria* in Revista Eletrônica da Associação Nacional dos Programas de Pos-graduação – Faculdade de Comunicação, Universidade Federal da Bahia (versão disponível somente em meio eletrônico - <http://www.facom.ufba.br/pretextos/alessandra.html>)
- MINISTÉRIO DA EDUCAÇÃO (1998) – *Diretrizes Curriculares de Cursos da Área de Computação e Informática* (versão eletrônica disponível em <http://www.mec.gov.br/Sesu/diretriz.shtm#diretrizes>)
- MULHOLLAND, P. (1998) *A Principled Approach to the Evaluation of SV: a Case Study in Prolog*. in Stasko, J., Domingue, J., Brown, M.H., Price, B.A. (org.) *Software Visualization: Programming as a Multimedia Experience*. Massachusetts Institute Technology Press, pp.439-451
- NAKAMURA, A. S., MASSI, C. D. B. (2001) *Um Estudo sobre Requisitos de Ferramentas que Auxiliam o Processo de Ensino-Aprendizagem em Lógica de Programação*. in Anais do XXI Congresso da Sociedade Brasileira de

Computação: IX Workshop Sobre Educação em Computação, Sociedade Brasileira de Computação, Fortaleza

OUDSHOORN, M.J., WIDJAJA, H.W, ELLERSHAW, S.K.. (1996) *Aspects and Taxonomy of Program Visualisation*. in EADES, P., ZHANG, K. (org.) *Software Visualization*, Chapter 1. World Scientific Press, Singapore (versão eletrônica disponível em <http://citeseer.nj.nec.com/oudshoorn96aspects.html>)

PAPERT, Seymour (1994) *A Máquina das Crianças* Artes Médicas, Porto Alegre, 1994, 210p.

PATTON, S. D. (1989) *The E-Machine: Supporting the Teaching of Program Execution Dynamics*. (Dissertação de Mestrado em Ciência da Computação) - Montana State University (versão eletrônica disponível em <http://esus.cs.montana.edu/~dynamlab/publications/theses/theses.html>)

PAXTON, J. T.; ROSS, ROCKFORD J.; STARKEY, J. (março de 1994). *A Methodology for Teaching an Integrated Computer Science Curriculum*. in SIGCSE'94 Bulletin, vol. 26, pp.1-5

PRICE, B. A., SMALL, I. S., and BAECKER, R. M. (1992). *A Taxonomy of Software Visualization*. in 25th Hawaii International Conference on System Sciences, volume II, IEEE Press, pp. 597-606.

PRICE, B. A.; BAECKER, R. M.; SMALL, I. S. (1993). *A Principled Taxonomy of Software Visualization*. in Journal of Visual Languages and Computing - 4 (3): pp.211-266

PRICE, B. A.; BAECKER, R. M.; SMALL, I. S. (1998). *An Introduction to Software Visualization*. in STASKO, J., DOMINGUE, J., BROWN, M.H., Price, B.A. (org.) *Software Visualization: Programming as a Multimedia Experience*. Massachusetts Institute Technology Press, pp.3-27

PRESSMAN, ROGER S. (1995). *Engenharia de Software*. 1.a Edição, Makron Books, 1056 p.

ROSENBERG, Jonathan B. (1996) *How Debuggers Work: Algorithms, Data Structure, and Architecture*. 1.a Edição, John Wiley Computer, 256 p.

- SALVETTI, D. D., BARBOSA, L. M. (1997) *Algoritmos*. 1.a Edição, Makron Books, 274 p.
- SANTIAGO, Sandra Helena Moreira (1997) *Breve Análise sobre a Adoção do Paradigma de "Rede" na Proposta de Multieducação* Trabalho apresentado na 20ª Reunião Anual da ANPED – Associação Nacional de Pós-Graduação e Pesquisa em Educação (versão eletrônica disponível em <http://www.ufrgs.br/faced/gtcurric/tr977.html>)
- SHACKELFORD, R.L. (1998). *Introduction to Computing and Algorithms*. 1.a Edição, Addison-Wesley, 412 p.
- SKIENA, S.S. (1998). *The Algorithm Design Manual*. 1.a Edição, Springer-Verlag, 486p.
- SOCIEDADE BRASILEIRA DE COMPUTAÇÃO – SBC (1999) *Currículo de Referência da SBC para Cursos de Graduação em Computação* (versão eletrônica disponível em <http://www.sbc.org.br/educacao/cr99.zip>)
- SPENCE, Robert (2001) *Information Visualization Using Vision to Think*, Morgan Kaufmann Publishers (ACM Press), Essex – England, 2001, p. 206
- STASKO, J. T., KRAEMER, E. (1993). *A methodology for Building Application-Specific Visualizations of Parallel Programs*. Journal of Parallel and Distributed Computing, 18 (2): pp. 258--264. (versão eletrônica disponível em <http://citeseer.nj.nec.com/stasko93methodology.html>)
- STASKO, J. T. (1996). *Using Student-Built Algorithm Animations as Learning Aids*. Graphics, Visualization, and Usability Center, Georgia Institute of Technology, Atlanta, GA, Technical Report GIT-GVU-96-19. (versão eletrônica disponível em <http://www.gvu.gatech.edu/gvu/softviz/publs/publs.html>)
- STASKO, J. T., LAWRENCE, A. (1998). *Empirically Assessing Algorithm Animations as Learning Aids*. in Stasko, J., Domingue, J., Brown, M.H., Price, B.A. (org.) *Software Visualization: Programming as a Multimedia Experience*. Massachusetts Institute Technology Press, pp.419-438.
- TERADA, R. (1991) *Desenvolvimento de Algoritmos e Estrutura de Dados*. São Paulo, 1.a Edição, Makron Books, 260 p.

- TIMÓTEO, D.M., BRASILEIRO, F. V. (1998). *A Formação e a Prática do Profissional de Programação: Discussão de Estratégias para a Aprendizagem Continuada*. in Anais do XVIII Congresso da Sociedade Brasileira de Computação: VI Workshop de Educação em Computação, Belo Horizonte, pp.505-520
- VALENTE, J.A. (1993a). *Diferentes Usos do Computador na Educação*. in VALENTE, J.A. (org.) *Computadores e Conhecimento: Repensando a Educação* Campinas, Editora da Unicamp, 1993. pp.1-23 (versão eletrônica disponível em <http://www.proinfo.gov.br/biblioteca/textos/txtie2doc.pdf>)
- VALENTE, J.A. (1993b). *Por Qué o Computador na Educação*. in VALENTE, J.A. (org.) *Computadores e Conhecimento: Repensando a Educação* Campinas, Editora da Unicamp, 1993. pp. 24-44 (versão eletrônica disponível em <http://www.proinfo.gov.br/biblioteca/textos/txtie9doc.pdf>)
- WIDJAJA, H., OUDSHOORN, M.J. (1997). *Concurrent Object Oriented Programming – A Visualisation Challenge*. in Proceedings of the Conference of Visual Data Exploration and Analysis IV, IS&T/SPIE Symposium on Electronic Imaging: Science and Technology, San Jose, California, pp. 310-321 (versão eletrônica disponível em <http://citeseer.nj.nec.com/widjaja97concurrent.html>)
- WARE, Colin (2000) *Information Visualization: Design for Perception* Morgan Kaufmann Publishers (Academic Press), 2000, 438p.

Anexo I - Tabela de Tokens

Os tokens são tratados como símbolos terminais na gramática da linguagem. Na maioria das linguagens de programação, as seguintes construções são tratadas como tokens: palavras reservadas, operadores, identificadores, constantes e símbolos de pontuação.

Identificador / Constante	
Identificador	-1
Caracter Especial	-2
Constante inteira	-3
Constante real	-4
Constante caracter	-5
Constante lógica	-6
Comentário	-7
Tabulação	-8
Espaço	-9

Palavras Reservadas	
<i>Algoritmo</i>	0
<i>Inicio</i>	1
<i>Escreva</i>	2
<i>Leia</i>	3
<i>Fimalgoritmo</i>	4
<i>Se</i>	5
<i>Então</i>	6
<i>Senao</i>	7
<i>Fimse</i>	8
<i>Para</i>	9
<i>De</i>	10
<i>Ate</i>	11
<i>Faca</i>	12
<i>Incremento</i>	13
<i>Fimpara</i>	14
<i>Repita</i>	15

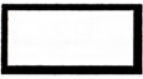
<i>Enquanto</i>	16
<i>Fimenquanto</i>	17
<i>Variavel</i>	18
<i>Tipo</i>	19
<i>Constante</i>	20
<i>Inteiro</i>	21
<i>Real</i>	22
<i>Caracter</i>	23
<i>Logico</i>	24
<i>Vetor</i>	25
<i>Registro</i>	26
<i>Funcao</i>	27
<i>Procedilmento</i>	28
<i>Val</i>	29
<i>Ref</i>	30
<i>E</i>	31
<i>Ou</i>	32
<i>Nao</i>	33
<i>Verdadeiro</i>	34
<i>Falso</i>	35
<i>Rq</i>	36
<i>Quad</i>	37
<i>Resto</i>	38
<i>Div</i>	39
<i>Int</i>	40
<i>Abs</i>	41
<i>Fimprocedimento</i>	42
<i>Fimfuncao</i>	43
<i>Fimregistro</i>	44
<i>Retorna</i>	45

Sinais de pontuação, operadores e outros	
<-	100
.	101
;	102
,	103
:	104
..	105
(	106
)	107
[	108
]	109
“	110
=	111
>	112
<	113
<>	114
<=	115
>=	116
+	117
-	118
*	119
/	120
{	121
}	122
\	123

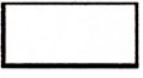


Anexo II – Árvore de Derivação Sintática da Linguagem Algorítmica

Simbologia adotada na especificação da árvore de derivação sintática:



Símbolo terminal



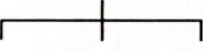
Símbolo não terminal com derivação abaixo



Símbolo não terminal opcional



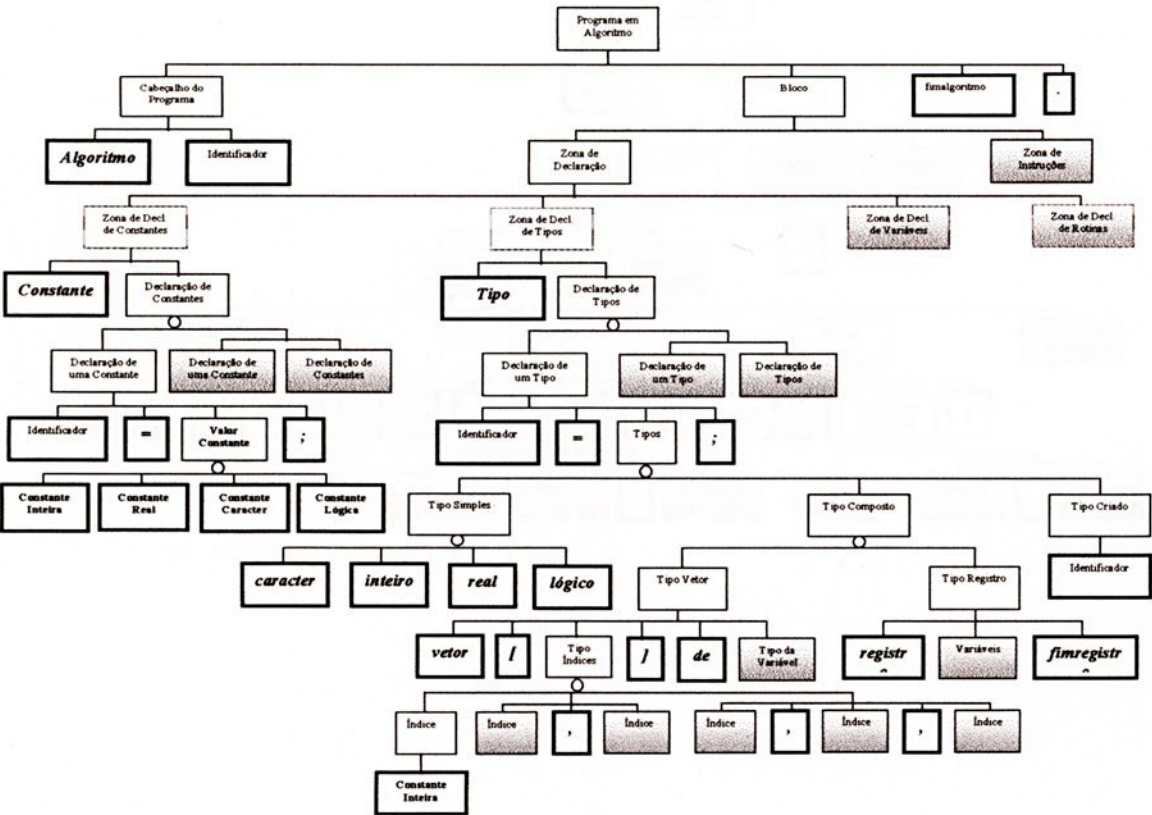
Símbolo não terminal já definido (derivado)

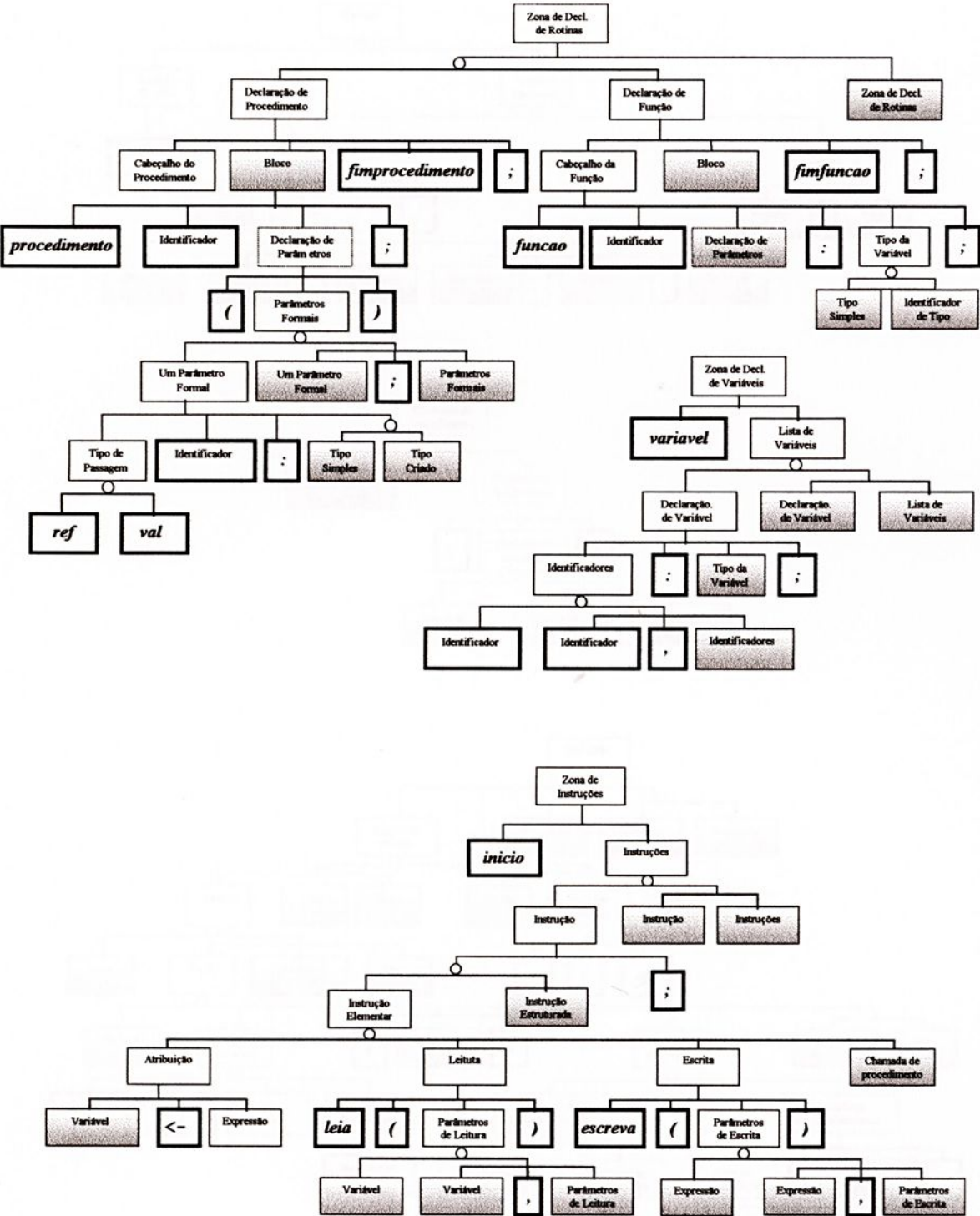


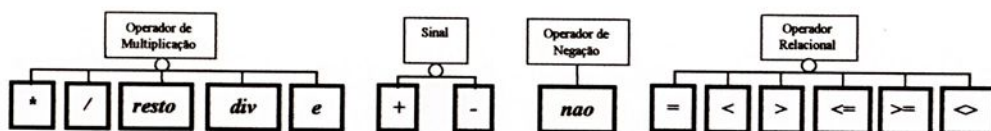
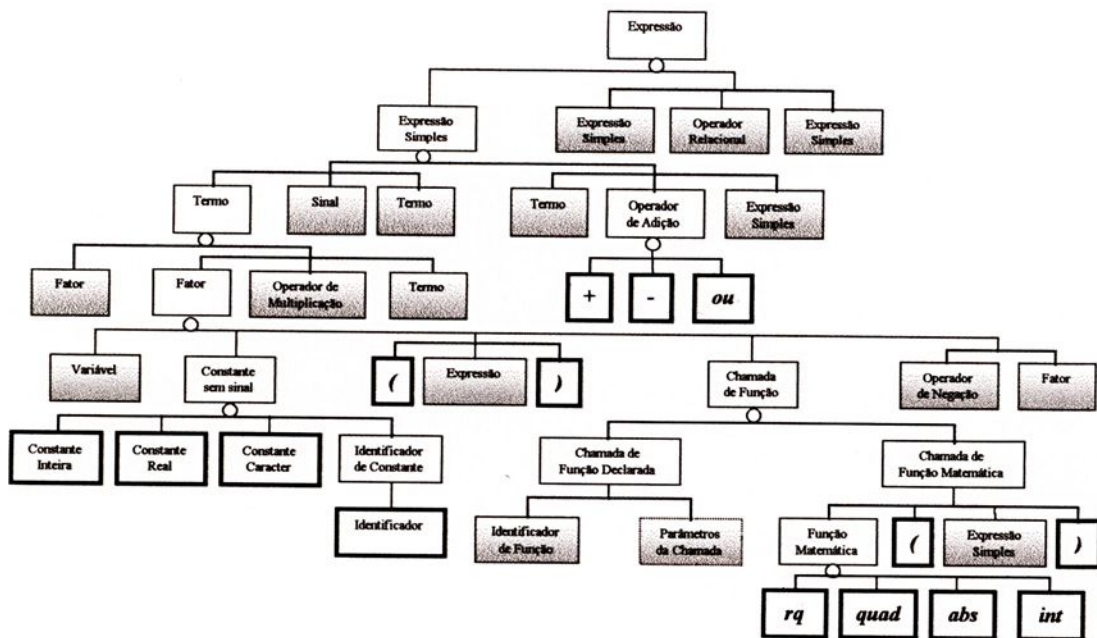
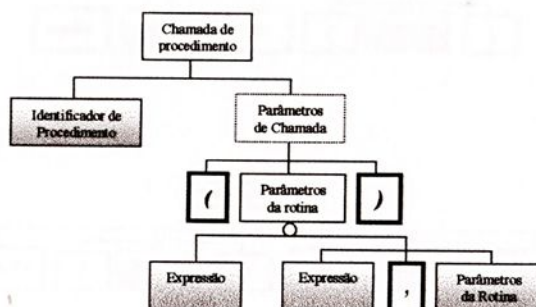
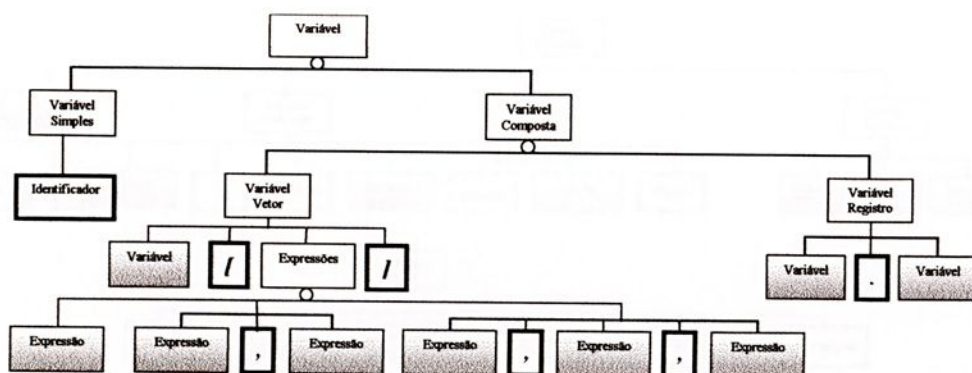
Derivação em uma sequência ordenada de símbolos

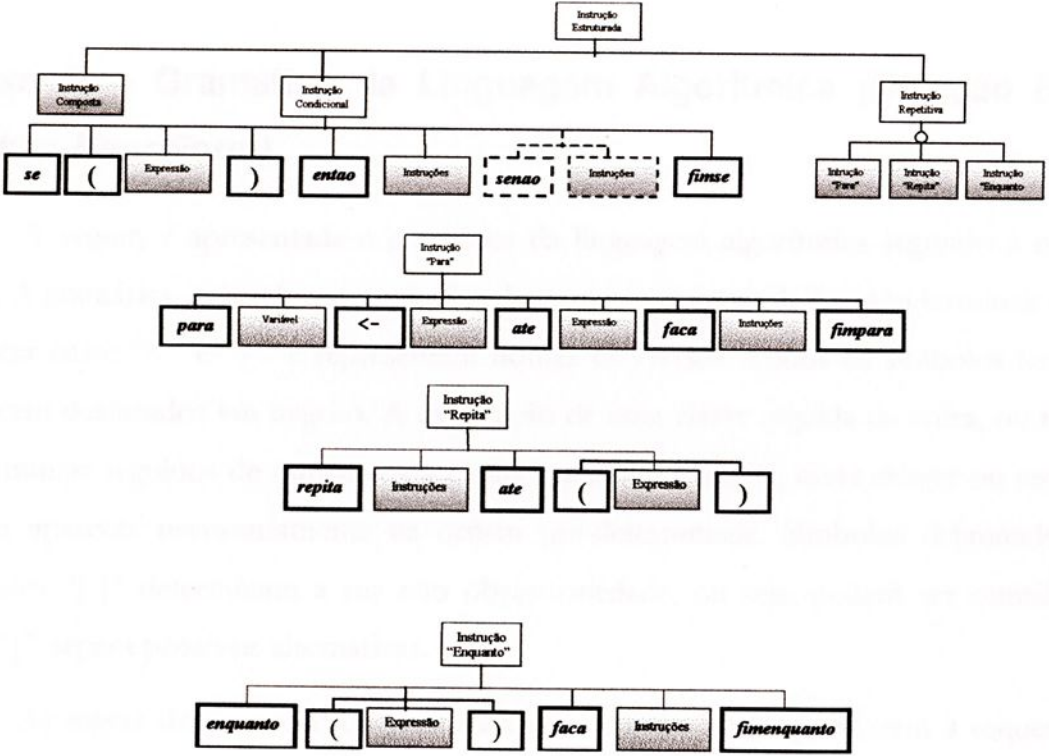


Derivação em uma alternativa entre os símbolos









Anexo III – Gramática da Linguagem Algorítmica (Notação BNF - Backus-Naur Form)

A seguir, é apresentada a gramática da linguagem algorítmica segundo a notação BNF. A gramática, segundo esta notação, determina que os símbolos não terminais devem aparecer entre “<” e “>” e representam nomes de classes. Todos os símbolos terminais aparecem destacados em negrito. A disposição de uma classe seguida da outra, ou mesmo de terminais seguidos de outros, indica uma seqüência, ou seja, essas classes ou terminais devem aparecer necessariamente na ordem pré-determinada. Símbolos delimitados por colchetes “[]” determinam a sua não obrigatoriedade, ou seja, podem ser omitidos. O sinal “|” separa possíveis alternativas.

As regras de produção da gramática em notação BNF apresentam à esquerda do símbolo “: =” um componente da linguagem (símbolo não terminal) e à sua direita uma descrição de seus componentes.

Os símbolos “{” e “}” foram acrescentados para otimizar o uso das regras de produção da gramática, objetivando uma redução substancial no seu número. Estes delimitam uma lista de alternativas dentro de uma seqüência.

A primeira regra de produção é apresentada a seguir:

- <programa em algoritmo> ::= <cabeçalho do algoritmo> <bloco> **fimalgoritmo** .

Nesta regra, a classe <programa em algoritmo> é composta pela seqüência de classes <cabeçalho do algoritmo> e <bloco> e pelos terminais “**fimalgoritmo**” seguido de “.”.

- <cabeçalho do algoritmo> ::= **algoritmo** **identificador**

O <cabeçalho do algoritmo> é composto pelos terminais **algoritmo** e **identificador**. Para cada classe deve haver uma regra de produção.

- <bloco> ::= <zona de declaração> <zona de instrução>
- <zona de declaração> ::= [<zona de declaração de constantes>] [<zona de declaração de tipos>] [<zona de declaração de variáveis>] [<zona de declaração de rotinas>]

- $\langle \text{zona de declaração de constantes} \rangle ::= \textit{constante} \langle \text{lista de constantes} \rangle$
- $\langle \text{lista de constantes} \rangle ::= \langle \text{declaração de uma constante} \rangle \mid \langle \text{declaração de uma constante} \rangle \langle \text{lista de constantes} \rangle$
- $\langle \text{declaração de uma constante} \rangle ::= \textit{identificador} = \langle \text{valor constante} \rangle;$
- $\langle \text{valor constante} \rangle ::= \textit{constante inteira} \mid \textit{constante real} \mid \textit{constante caracter}$
- $\langle \text{zona de declaração de tipos} \rangle ::= \textit{tipo} \langle \text{lista de tipos} \rangle$
- $\langle \text{lista de tipos} \rangle ::= \langle \text{declaração de um tipo} \rangle \mid \langle \text{declaração de um tipo} \rangle \langle \text{lista de tipos} \rangle$
- $\langle \text{declaração de um tipo} \rangle ::= \textit{identificador} = \langle \text{tipos} \rangle;$
- $\langle \text{tipos} \rangle ::= \langle \text{tipo simples} \rangle \mid \langle \text{tipo composto} \rangle \mid \langle \text{tipo criado} \rangle$
- $\langle \text{tipo simples} \rangle ::= \textit{inteiro} \mid \textit{real} \mid \textit{caracter} \mid \textit{lógico}$
- $\langle \text{tipo composto} \rangle ::= \langle \text{tipo vetor} \rangle \mid \langle \text{tipo registro} \rangle$
- $\langle \text{tipo vetor} \rangle ::= \textit{vetor} [\langle \text{tipo índice} \rangle] \textit{de} \langle \text{tipo da variável} \rangle$
- $\langle \text{tipo índice} \rangle ::= \langle \text{índice} \rangle \mid \langle \text{índice} \rangle, \langle \text{índice} \rangle \mid \langle \text{índice} \rangle, \langle \text{índice} \rangle, \langle \text{índice} \rangle$
- $\langle \text{índice} \rangle ::= \textit{constante inteira} \mid \textit{identificador}$
- $\langle \text{tipo registro} \rangle ::= \textit{registro} \langle \text{lista de variáveis} \rangle \textit{fimregistro}$
- $\langle \text{zona de declaração de variáveis} \rangle ::= \textit{variavel} \langle \text{lista de variáveis} \rangle$
- $\langle \text{lista de variáveis} \rangle ::= \langle \text{declaração de variáveis} \rangle \mid \langle \text{declaração de variáveis} \rangle \langle \text{lista de variáveis} \rangle$
- $\langle \text{declaração de variáveis} \rangle ::= \langle \text{identificadores} \rangle : \langle \text{tipo da variável} \rangle;$
- $\langle \text{identificadores} \rangle ::= \textit{identificador} \mid \textit{identificador}, \langle \text{identificadores} \rangle$
- $\langle \text{tipo da variável} \rangle ::= \langle \text{tipo simples} \rangle \mid \langle \text{tipo criado} \rangle$
- $\langle \text{tipo criado} \rangle ::= \textit{identificador}$
- $\langle \text{zona de declaração de rotinas} \rangle ::= \langle \text{declaração de rotina} \rangle$

- <declaração de rotina>::= { <declaração de procedimento> | <declaração de função> }
[<declaração de rotina>]
- <declaração de procedimento>::= <cabeçalho do procedimento> <bloco>
fimprocedimento;
- <cabeçalho do procedimento>::= **procedimento identificador** [<declaração de
parâmetros>] ;
- <declaração de função>::= <cabeçalho da função> <bloco> **fimfuncao;**
- <cabeçalho da função>::= **funcao identificador** [<declaração de parâmetros>] :
<tipo da variável>;
- <declaração de parâmetros>::= (<parâmetros formais>)
- <parâmetros formais>::= <um parâmetro formal> | <um parâmetro formal> ,
<parâmetros formais>
- <um parâmetro formal>::= <tipo de passagem> **identificador** : <tipo da variável>
- <zona de instruções>::= **inicio** <instruções>
- <instruções>::= <instrução> | <instrução> <instruções>
- <instrução>::= { <instrução elementar> | <instrução estruturada> } ;
- <instrução elementar>::= <atribuição> | <leitura> | <escrita> | <chamada de
procedimento>
- <atribuição>::= <variável> <- <expressão>
- <leitura>::= **leia** (<parâmetros de leitura>)
- <parâmetros de leitura>::= <variável> | <variável> , <parâmetros de leitura>
- <escrita>::= **escreva** (<parâmetros de escrita>)
- <parâmetros de escrita>::= <expressão> | <expressão> , <parâmetros de escrita>
- <chamada de procedimento>::= **identificador** [(<parâmetros da rotina>)]
- <parâmetros da rotina>::= <expressão> | <expressão> , <parâmetros da rotina>

- $\langle \text{instrução estruturada} \rangle ::= \langle \text{instrução condicional} \rangle \mid \langle \text{instrução de repetição} \rangle$
- $\langle \text{instrução condicional} \rangle ::= \text{se } (\langle \text{expressão} \rangle) \text{ então } \langle \text{instruções} \rangle \text{ [se não } \langle \text{instruções} \rangle] \text{ fimse}$
- $\langle \text{instrução de repetição} \rangle ::= \langle \text{instrução "para"} \rangle \mid \langle \text{instrução "repita"} \rangle \mid \langle \text{instrução "enquanto"} \rangle$
- $\langle \text{instrução "para"} \rangle ::= \text{para } \langle \text{variável} \rangle \text{ <- } \langle \text{expressão} \rangle \text{ ate } \langle \text{expressão} \rangle \text{ [incremento } \langle \text{expressão} \rangle] \text{ faça } \langle \text{instruções} \rangle \text{ fimpara}$
- $\langle \text{instrução "repita"} \rangle ::= \text{repita } \langle \text{instruções} \rangle \text{ ate } (\langle \text{expressão} \rangle)$
- $\langle \text{instrução "enquanto"} \rangle ::= \text{enquanto } (\langle \text{expressão} \rangle) \langle \text{instruções} \rangle \text{ fimenquanto}$
- $\langle \text{expressão} \rangle ::= \langle \text{expressão simples} \rangle \mid \langle \text{expressão simples} \rangle \langle \text{operador relacional} \rangle \langle \text{expressão simples} \rangle$
- $\langle \text{expressão simples} \rangle ::= \langle \text{termo} \rangle \langle \text{sinal} \rangle \langle \text{termo} \rangle \mid \langle \text{termo} \rangle \langle \text{operador de adição} \rangle \langle \text{expressão simples} \rangle$
- $\langle \text{sinal} \rangle ::= + \mid -$
- $\langle \text{operador de adição} \rangle ::= + \mid - \mid \text{ou}$
- $\langle \text{termo} \rangle ::= \{ \langle \text{fator} \rangle \mid \langle \text{fator} \rangle \langle \text{operador de multiplicação} \rangle \langle \text{termo} \rangle \}$
- $\langle \text{operador de multiplicação} \rangle ::= * \mid / \mid \text{resto} \mid \text{div} \mid e$
- $\langle \text{fator} \rangle ::= \langle \text{variável} \rangle \mid \langle \text{constante sem sinal} \rangle \mid (\langle \text{expressão} \rangle) \mid \langle \text{chamada de função} \rangle \mid \langle \text{operador de negação} \rangle \langle \text{fator} \rangle$
- $\langle \text{constante sem sinal} \rangle ::= \text{constante real} \mid \text{constante inteira} \mid \text{constante caracter} \mid \text{identificador}$
- $\langle \text{chamada de função} \rangle ::= \langle \text{chamada de função declarada} \rangle \mid \langle \text{chamada de função matemática} \rangle$
- $\langle \text{chamada de função declarada} \rangle ::= \text{identificador } [(\langle \text{parâmetros da rotina} \rangle)]$
- $\langle \text{chamada de função matemática} \rangle ::= \langle \text{função matemática} \rangle (\langle \text{expressão simples} \rangle)$

- $\langle \text{operador de negação} \rangle ::= \text{nao}$
- $\langle \text{operador relacional} \rangle ::= \{ = | > | < | >= | <= | <> \}$
- $\langle \text{variável} \rangle ::= \langle \text{variável simples} \rangle | \langle \text{variável composta} \rangle$
- $\langle \text{variável simples} \rangle ::= \text{identificador}$
- $\langle \text{variável composta} \rangle ::= \langle \text{variável vetor} \rangle | \langle \text{variável registro} \rangle$
- $\langle \text{variável vetor} \rangle ::= \langle \text{variável} \rangle [\langle \text{índices do vetor} \rangle]$
- $\langle \text{variável registro} \rangle ::= \langle \text{variável} \rangle . \langle \text{variável} \rangle$
- $\langle \text{índices do vetor} \rangle ::= \langle \text{expressão} \rangle | \langle \text{expressão} \rangle , \langle \text{expressão} \rangle | \langle \text{expressão} \rangle , \langle \text{expressão} \rangle , \langle \text{expressão} \rangle$

Anexo IV – Conteúdo Programático da Disciplina Construção de Algoritmos


UNIVERSIDADE DE MARÍLIA

PROGRAMA ANALÍTICO DE DISCIPLINA

Período: 1º./2001

1. IDENTIFICAÇÃO

FACULDADE:	FCET – Faculdade de Ciências Exatas e Tecnológicas				
CURSO:	3223 – Ciência da Computação (Diurno)			TERMO:	1º
DISCIPLINA:	140260 – Construção de Algoritmos				
CARGA HORÁRIA TOTAL:	68	CARGA HORÁRIA TOTAL TEÓRICA:	44	CARGA HORÁRIA TOTAL PRÁTICA:	24
PRÉ-REQUISITO:					

2. OBJETIVOS

Fornecer técnicas de desenvolvimento da lógica de programação, criar habilidades para resolução de problemas lógicos e instituir conceitos que permitam o aprimoramento do estilo e da qualidade no processo de desenvolvimento de algoritmos.

3. EMENTA

Conceitos Básicos de Lógica de Programação. Tipos de Dados Primitivos. Variáveis e Constantes. Regras para Declaração de Identificadores. Estrutura Genérica de um Algoritmo. Fluxo de Dados. Atribuição de valores. Operadores matemáticos, relacionais e lógicos. Expressões Lógicas e Aritméticas. Entrada e Saída de Dados. Teste de Mesa. Tabelas Verdade. Blocos de Execução. Estruturas de Decisão. Estruturas de Repetição. Estruturas de Dados. Modularização. Estilo e Qualidade de Programação.

4. DISCRIMINAÇÃO DO CONTEÚDO PROGRAMÁTICO

Conceitos Básicos de Lógica de Programação

- Definição de Algoritmo
- Importância do Algoritmo para o Curso
 - Exemplos de Linguagens de Programação
- Noções de Lógica
- Lógica do Dia-a-Dia
- Exemplo de Algoritmos
 - Troca de Lâmpada
 - Receita de Bolo
- Exercício: Escrever um algoritmo retirado do dia-a-dia

Fluxo de Dados

- Definição de fluxo de dados
- Importância da Utilização do Fluxo de dados
- Caixas gráficas utilizadas para a construção do fluxo de dados
- Exemplos de algoritmos utilizando fluxo de dados



Tipos de Dados Primitivos

- Definição dos valores primitivos
- Definição dos tipos primitivos
- Exemplos de Valores para cada tipo primitivo
- Declaração dos tipos inteiro, lógico, caracter e real.
- Exercícios para Identificação de valores

Regras para Declaração de Identificadores

- Definição das regras para declaração de variáveis e constantes
- Exemplos de declarações corretas e incorretas
- Exercícios utilizando as regras de declaração e correção dos exercícios do item anterior

Variáveis e Constantes

- Noções sobre memória do computador
- Definição de Variáveis e Constantes
- Sintaxe para declaração das variáveis e constantes
- Exemplos de declarações
- Exercícios envolvendo declarações de variáveis e constantes

Atribuição de Valores

- Definição e sintaxe de atribuição de valores
- Significado físico da atribuição
- Erros de atribuição relacionado à tipos de dados e variáveis incompatíveis
- Utilização de comentários
- Exercícios

Operadores Matemáticos, Relacionais e Lógicos

- Definição dos operadores matemáticos, relacionais e lógicos
- Apresentação dos operadores
 - Adição, Subtração, Divisão e Multiplicação
 - Radiciação e Exponenciação
 - Operadores E, OU e NÃO
 - Tabelas Verdade para os operadores Lógicos
 - Operadores relacionais maior que, menor que, maior e igual, menor e igual e igual
- Prioridade na execução
- Utilização de parêntesis

Expressões Lógicas e Aritméticas

- Definição de expressões lógicas Aritméticas
- Exemplos de expressões envolvendo todos os operadores
- Exercícios envolvendo expressões lógicas e aritméticas

Entrada e Saída de Dados

- Conceito fundamental do processamento de dados
 - Entrada de dados, Processamento e Saída de dados
 - Dispositivos de entrada e saída de dados
 - Dispositivos de entrada e saída de dados padrões
 - Comandos de Entrada e Saída de dados
 - Leia (variável)
 - Escreva (expressão)
 - Exemplos de algoritmos utilizando entrada e saída de dados
 - Exercícios propostos sobre entrada e saída de dados
-



Estrutura Genérica de um Algoritmo

Representação genérica de um algoritmo

Cabeçalho

Área de declarações

Corpo principal

A importância da Indentação

Exemplo de um algoritmo utilizando os conceitos de estrutura genérica

Blocos de execução

Definição de bloco de execução

Como delimitar um bloco de execução

Como algumas linguagens delimitam um bloco de execução

Exemplo utilizando um algoritmo

Exercícios

Teste de Mesa

Definição de teste de mesa

Importância do teste de mesa

Como e quando realizar o teste de mesa

Exemplo de teste de mesa

Exercícios incluindo fluxogramas e teste de mesa

Estruturas de Decisão (Seleção)

Estrutura de Seleção Simples (Se ... então...fimse)

Definição da estrutura

Descrição do funcionamento

Quando utilizar

Exemplos de utilização desta estrutura

Estrutura de Seleção Composta (Se ... então...senão ...fimse)

Definição da estrutura

Descrição do funcionamento

Quando utilizar

Exemplos de utilização desta estrutura

Estrutura de Seleção Encadeada

Definição da estrutura

Descrição do funcionamento

Quando utilizar

Exemplos de utilização desta estrutura

Exercícios envolvendo estruturas de seleção

Estrutura de Seleção de Múltipla Escolha (Escolha ... caso ... fimsecolha)

Definição da estrutura

Descrição do funcionamento

Quando utilizar

Exemplos de utilização desta estrutura

Exercícios envolvendo estruturas de seleção de múltipla escolha

Estruturas de Repetição

Estrutura de Repetição com Teste no Início (Enquanto ... faça ...fimenquanto)

Definição da estrutura

Descrição do funcionamento

Quando utilizar

Exemplos de utilização desta estrutura

Exercícios



Estrutura de Repetição com Teste no Final (Repita... até)

- Definição da estrutura
- Descrição do funcionamento
- Quando utilizar
- Exemplos de utilização desta estrutura
- Exercícios

Estrutura de Repetição com Variável de Controle

- Definição da estrutura
- Descrição do funcionamento
- Quando utilizar
- Exemplos de utilização desta estrutura
- Exercícios

Comparação entre as estruturas de repetição

- Exercícios

Estrutura de Dados

- Definição de estruturas de dados
- Necessidade de organização dos dados
- Exemplos de organização de dados do dia-a-dia
- Variáveis Compostas Homogêneas

Variáveis Compostas Unidimensionais (vetores)

- Definição
- Significado Físico da estrutura (memória)
- Sintaxe para declaração
- Forma de Utilização
- Exemplos de declaração e utilização
- Exercícios

Variáveis Compostas Multidimensionais (matrizes)

- Definição
- Significado Físico da estrutura (memória)
- Sintaxe para declaração
- Forma de Utilização
- Exemplos de declaração e utilização
- Exercícios

Variáveis Compostas Heterogêneas**Registros**

- Definição
- Conceito de inclusão de tipos novos
- Utilização dos registros no dia-a-dia
- Significado Físico da estrutura (memória)
- Sintaxe para declaração
- Forma de Utilização
- Exemplos de declaração e utilização
- Exercícios

Modularização

- Definição e importância da modularização
- Exemplos práticos de modularização
- Vantagens oferecidas pela modularização
- Procedimentos
 - Definição
 - Declaração de procedimentos
 - Exemplos de declarações em linguagens de programação
 - Conceito de parâmetros
 - Escopo de uma variável

**UNIVERSIDADE DE MARÍLIA****Funções**

Definição e importância da utilização
Declaração de funções
Diferença entre procedimentos e funções
Exemplos de declarações em linguagens de programação
Exemplos de declarações de funções em algoritmo
Exercícios

Estilo e Qualidade de Programação

Ética e estilo de programação
Otimização da memória
Replicação de variáveis
Variáveis não aproveitadas
Replicação de código
Utilização de Procedimentos e Funções
Indentação e Inserção de Comentários
Escolha da Linguagem de Programação
Implementação Top-Down e Bottom-up

5. METODOLOGIA DE ENSINO

Utilização de recursos audio-visuais, aulas teóricas, livro texto, listas de exercícios e software para execução dos algoritmos desenvolvidos pelos alunos.

6. SISTEMAS DE AVALIAÇÃO

Duas avaliações por bimestre, sendo a média bimestral calculada através da média aritmética dessas avaliações.

7. BIBLIOGRAFIA

FORBELLONE, André Luiz Villar; Eberspacher, Henri Frederico; "*Lógica de Programação*" A *Construção de Algoritmos e Estruturas de Dados*; 2ª Edição; Makron Books; 1999.

TREMBLAY, E.; "*Ciência dos Computadores: Uma abordagem algorítmica*"; Makron Books; 1986.

FARRER Harry; "*Algoritmos Estruturados*"; 3ª Edição; Editora JC; 1999.

SALVETTI, Dirceu Douglas; BARBOSA, Lisbete Madsen; "*Algoritmos*"; Makron Books; 1998.

8. APROVAÇÃO

MARÍLIA, _____ DE _____ DE 2001

Prof. Adriano Sunao Nakamura

Assinatura do Coordenador

Assinatura do Diretor



Anexo V – Questionário submetido aos alunos que ingressaram em 2000 e 2001 nos cursos de Bacharelado em Ciência da Computação e Sistemas de Informação / Tabulação das Respostas

Caro(a) Aluno(a)

Este questionário tem como objetivo coletar dados para uma pesquisa que venho desenvolvendo sobre o ensino da lógica de programação. Este levantamento possui propósitos unicamente acadêmicos/científicos. Os dados obtidos servirão de subsídio para traçar o perfil dos ingressos na faculdade e avaliar o ensino da disciplina Construção de Algoritmos. Sua contribuição será de suma importância. Não há necessidade de identificar-se.

Agradeço sua atenção e colaboração
Prof. Adriano Sunao Nakamura

1. Sexo: ☐ Masculino ☐ Feminino
2. Idade: ____ anos
3. Estado civil: a[☐] solteiro b[☐] casado c[☐] outro _____
4. Em que rede de ensino estudou:
1º grau a[☐] pública b[☐] privada c[☐] ambas
2º grau a[☐] pública b[☐] privada c[☐] ambas
5. Qual a formação que obteve no 2º grau:
a[☐] Científico/Normal
b[☐] técnico em Processamento de Dados
c[☐] técnico em Contabilidade
d[☐] Magistério
e[☐] outro _____
6. Você trabalha? ☐ Não ☐ Sim, ____ horas por dia
7. Quem arca com os custos de seu estudo?
a[☐] você mesmo
b[☐] sua família
c[☐] a empresa onde trabalha, que subsidia c1[☐] integral c2[☐] parcialmente
d[☐] financiamento via FIES ou Bolsa (Unimar)
8. Sua residência fica em
a[☐] Marília (residência fixa)
b[☐] Marília, temporariamente para os estudos
c[☐] municípios vizinhos de Marília – qual a distância até a sua cidade? ____ Km
9. Se sua residência fixa não é em Marília, você vai para casa:
a[☐] todos os dias
b[☐] nos fins de semana
c[☐] somente em feriados e férias escolares
10. Você utiliza diariamente o computador? ☐ Não ☐ Sim, ____ horas por dia
11. Qual o meio mais utilizado para manter-se informado? (assinale uma das alternativas)
a[☐] livros b[☐] jornais c[☐] revistas d[☐] rádio e[☐] televisão f[☐] Internet
12. Em conversa informal com os amigos, que assunto ocorre com maior frequência? (assinale uma das alternativas)
a[☐] computadores b[☐] estudos c[☐] trabalho d[☐] festas
f[☐] outros assuntos _____
13. Em qual curso está matriculado? a[☐] C. Computação b[☐] S. Informação c[☐] P. Dados
14. Em que ano ingressou no curso? _____
15. Quantos vestibulares você prestou até ingressar neste curso?
a[☐] um b[☐] dois c[☐] mais de dois
16. Em que período está matriculado? a[☐] diurno b[☐] noturno

17. O que o levou a escolher o curso em questão? (assinale uma das alternativas)
- a[] influência de colegas
 - b[] influência dos pais
 - c[] influência de professores
 - d[] necessidades relativas ao trabalho que desempenha
 - e[] afinidade com a área de informática
 - f[] expectativa de ingressar facilmente no mercado de trabalho
18. Você está fazendo (só fez) a disciplina Construção de Algoritmos pela primeira vez?
- [] Sim [] Não
- Na última vez em que se matriculou nesta disciplina, você:
- a[] abandonou no início
 - b[] abandonou no meio
 - c[] foi até o fim
19. Você tem encontrado (encontrou) alguma dificuldade na disciplina Construção de Algoritmos?
- [] Sim [] Não
- Qual o principal motivo? (assinale uma das alternativas)
- a[] falta didática/domínio ao professor
 - b[] não se esforçou o suficiente
 - c[] o tempo dedicado a cada assunto durante a aula é muito reduzido
 - d[] o material adotado no curso é de qualidade duvidosa
 - e[] deficiência no ensino obtido durante os 1º e 2º graus
 - f[] outros _____
20. Durante o decorrer do primeiro semestre houve disciplinas em que você encontrou maior dificuldade que a de Construção de Algoritmo?
- [] Não [] Sim - Quais? _____
21. Qual o grau de importância que você atribui a disciplina Construção de Algoritmos para o curso? (assinale uma das alternativas)
- a[] é uma das principais disciplinas do curso; serve de base para a maioria das outras disciplinas
 - b[] é uma disciplina importante como qualquer outra disciplina da área técnica
 - c[] poderia ser substituído por uma disciplina que ensinasse a programar utilizando uma linguagem de programação executável
 - d[] é descartável, uma vez que o seu conteúdo é abordado em outras disciplinas que envolvem programação.
22. Sente motivado a (pretende) continuar o curso?
- [] Sim [] Não
- Qual o principal motivo? (assinale uma das alternativas)
- a[] desmotivado com as notas baixas nas disciplinas
 - b[] não se identificou com o curso
 - c[] a instituição não correspondeu com suas expectativas
 - d[] problemas financeiros
 - e[] outros _____
23. (somente para alunos que atualmente cursam a disciplina Construção de Algoritmos) Na sua opinião, o uso da ferramenta AVA (Ambiente para Desenvolvimento e Visualização de Algoritmo) em Construção de Algoritmos: (assinale uma das alternativas)
- a[] tem apresentado um papel indispensável no acompanhamento da disciplina
 - b[] serve como uma ferramenta de auxílio no teste do algoritmo
 - c[] é irrelevante; o desempenho na disciplina seria o mesmo sem a sua utilização
 - d[] dificulta a aprendizagem, devido aos detalhes técnicos de sua utilização.

Tabulação das Respostas

Sexo	Qtde	%
Masculino	121	78,6
Feminino	33	21,4

Dados sobre Idade		
Idade Média	21,88	
Entre 18 a 22	114	(74,0%)
Maior idade	37	
Menor idade	17	

Estado Civil	Qtde	%
Solteiro	136	88,3%
Casado	16	10,4%
Outros	2	1,3%

Rede	1.o Grau		2.o Grau	
	Qtde	%	Qtde	%
Pública	113	73,4%	90	58,4%
Privada	33	21,4%	41	26,6%
Ambas	8	5,2%	23	15,0%

Formação 2.o Grau	Qtde	%
Científico/Normal	120	77,9
Tec. Processamento	15	9,8
Tec. Contabilidade	3	2,0
Magistério	5	3,2
Outros	11	7,1

Trabalha	Qtde	%
Sim	84	54,5
Não	70	45,5

Custos com o curso	Qtde	%
Você	41	26,6
Família	87	56,5
Empresa onde trabalha	7	4,5
FIES / bolsa Unimar	19	12,4

Residência	Qtde	%
Marília (nativo)	57	37,2
Marília (temporário.)	44	28,8
Cidades Vizinhas	52	34,0

Viaja p/ cidade Natal	Qtde	%
Todo dia	52	54,2
Nos fim semana	14	14,6
Nos feriados	30	31,2

Usa computador	Qtde	%
Sim	120	77,9
Não	34	22,1
Media horária de uso	4,47	

Meios de informação	Qtde	%
Livros	2	1,3
Jornais	26	17,1
Revistas	7	4,6
rádio	1	0,6
televisão	67	44,2
internet	49	32,2

Assuntos em conversas	Qtde	%
computadores	42	27,3
estudo	43	27,9
trabalho	21	13,7
festas	23	14,9
outros	25	16,2

Curso	Qtde	%
CC	115	74,7
SI	39	25,3

Vestibulares	Qtde	%
Somente 1	108	70,1
2	24	15,6
Mais de 2	22	14,3

Período	Qtde	%
diurno	41	26,6
noturno	113	73,4

Escolha do Curso	Qtde	%
Colegas	5	3,3
Pais	3	2,0
Professores	0	0,0
Imposição do Trabalho	19	12,6
Afinidade com a Informática	75	49,7
Expectativa de Emprego	49	32,4

Sobre a Disciplina Construção de Algoritmos

<i>Cursando a disciplina</i>	<i>Qtde</i>	<i>%</i>
1.a vez	133	88,7
mais vezes	17	11,3
abandonou no início	1	5,9
abandonou no meio	0	0,0
foi até o fim	16	94,1

<i>Dificuldade na Disciplina</i>	<i>Qtde</i>	<i>%</i>
Sim	100	65,3
Não	53	34,7
<i>Motivo apontado p/ a dificuldade</i>	<i>Qtde</i>	<i>%</i>
Professor não tem didática / domínio	5	5,2
Não se esforçou o suficiente	31	32,0
Carga horária é insuficiente	30	30,9
Falhas no material didático	3	3,1
Deficiência herdada do ensino médio	18	18,5
outros	10	10,3

<i>Dificuldades em outras disciplinas</i>	<i>Qtde</i>	<i>%</i>
sim	64	42,9
não	85	57,1
<i>Disciplinas mais difíceis</i>	<i>Qtde</i>	<i>%</i>
Cálculo Diferencial e Integral	28	45,2
Comunicação e Expressão	10	16,1
Introdução a Ciência da Computação	1	1,6
Geometria Analítica e Álgebra Linear	15	24,2
Matemática Discreta	8	12,9

<i>Grau de Importância da Disciplina</i>	<i>Qtde</i>	<i>%</i>
É principal	120	80,5
É igual às outras disciplinhas técnicas	19	12,8
Pode ser substituído por linguagem de prog.	9	6,0
É descartável	1	0,7

<i>Motivação</i>		
Sim	127	85,8
Não	21	14,2
<i>Causa da desmotivação</i>	<i>Qtde</i>	<i>%</i>
Notas baixas	9	48,9
Não se identificou c/ o curso	2	9,5
Não atingiu às expectativas	1	4,8
Problemas financeiros	7	33,3
Outros	2	9,5

<i>Sobre o uso da ferramenta AVA</i>	<i>Qtde</i>	<i>%</i>
É indispensável	21	28,0
É uma mera ferramenta	50	66,7
Seu uso é irrelevante	1	1,3
Dificulta a aprendizagem	3	4,0

Anexo VI – Descrição da Linguagem Algorítmica

Este texto foi utilizado como parte do material didático distribuído aos alunos do 1.o período letivo de 2001.

1. Tipos Primitivos

Num sistema de processamento de dados, encontramos três fases distintas: entrada, processamento e saída. Em cada uma destas fases encontramos um elemento peculiar sem o qual não se justificaria a existência do sistema em si: os dados. Esses elementos devem ser fornecidos como entrada; ao serem processados, sofrem transformações e resultados são exibidos como saída.

Os dados manipulados podem ser classificados de diversas formas, dependendo de sua natureza. Para a linguagem algorítmica em questão, inicialmente, os classificaremos em quatro categorias que denominamos de tipos primitivos:

- **Inteiro:** tipo que determina todo e qualquer dado que possa ser quantificado através de número que pertença ao conjunto dos números inteiros. Exemplos de dados do tipo inteiro: número de filhos (ele tem 3 filhos), número de carros (ele comprou 2 carros), número de degraus de uma escada (a escada possui 15 degraus), ...
- **Real:** tipo que determina todo e qualquer dado que possa ser quantificado através de número que pertença ao conjunto dos números reais. Exemplo de dado do tipo real: altura (minha altura é 1,75 metros), peso (comprei 1,5 kilos de carne), preço (o quilo da carne custa 4,50 reais), saldo bancário (meu saldo era de - 547,21 reais)...
- **Caracter:** tipo que determina todo e qualquer dado que possa ser representado por conjunto de caracteres alfanuméricos, ou seja, qualquer combinação de caracteres numéricos (0..9), caracteres alfabéticos (a .. z, A ..Z) e caracteres especiais (exemplo: +, -, *, [, @, %, \$, *, #, ...). Exemplo de dados tipo caracter: nome do cliente (o cliente *Asdrúbal Marcondes* realizou uma compra ontem), endereço eletrônico (enviei uma mensagem para o endereço eletrônico *novo-mail@internet.com*),
- **Lógico:** tipo que determina valores de capacidade binária, ou seja, assumem somente dois valores possíveis: Verdadeiro ou Falso. Exemplos de dados do tipo lógico: expressões (a expressão $2 > 3$ é *Falsa*), casado (pode assumir *Verdadeiro* ou *Falso* dependendo do estado civil). - veja na seção expressões lógicas a utilidade dos tipos lógicos.

2. Variáveis e Constantes

A representação dos dados num algoritmo pode ser feita basicamente de duas formas: constante e variável. Tanto a constante como a variável podem ser encaradas como uma caixa ou uma célula devidamente identificada, onde os dados são armazenados. Sempre que for necessário acessar determinado dado, faz menção ao nome da constante ou variável (veja seção 3.1 – Identificadores). Num computador, uma constante ou variável está associada a um endereço de memória onde os valores são armazenados.

O que caracteriza uma variável é o simples fato de se poder alterar o valor nela armazenado durante a execução de um algoritmo. Por exemplo, o saldo bancário pode assumir um valor no início do algoritmo e, durante a sua execução, podem ser realizados vários créditos e débitos; como resultado dessas operações, o valor armazenado na variável em questão vai tendo o seu valor armazenado alterado. Outros exemplos de dados variáveis: nome do cliente, preço do produto, quantidade do produto,...

Já uma constante faz exatamente o que o próprio nome sugere. Trata-se de um valor que não é variável, ou seja, permanece o mesmo durante toda a extensão do algoritmo. Exemplos de dados constantes: FALSO, "Proibido fumar!", 3.1415,...

2.1. Identificadores

Identificador é simplesmente uma terminologia usada para nomear objetos (variáveis, constantes, tipos definidos, procedimentos funções,...) num algoritmo. Um identificador único deve ser designado para cada objeto a ser utilizado num algoritmo, de forma a evitar ambigüidade. Uma das principais características do identificador é ser descritivo, ou seja, seu nome deve descrever o objeto ao qual está relacionado. Exemplo: o identificador *nome_cliente* é altamente descritivo quando estiver relacionado ao nome do cliente, da mesma forma, *preco_prod* para preço do produto e *calcular_media* para uma rotina que calcula a média.

Algumas restrições são impostas à formação do identificador:

- Deve começar com um caracter alfabético (a..z, A ..Z, exceto ç, e caracteres acentuados);
- A partir do segundo caracter podem ser utilizados em qualquer ordem e alternadamente caracteres alfabéticos (a..z, A ..Z, exceto ç e caracteres acentuados) e caracteres numéricos (0..9);
- Não é permitido o uso de caracteres especiais (+, -, *, [, @, %, \$, *, #,...), qualquer tipo de pontuação nem espaços em branco.
- Como separador, pode ser utilizado o caracter "_" (exemplo: nome_do_cliente).

Exemplos de identificadores inválidos: 123, nome-cliente, nº, R\$, nota 1, 4X, RU@.

Exemplos de identificadores válidos: um_dois_tres, NomeCliente, NRO, Nota_1, QuatroX, rua.



2.2. Declaração de Variáveis e Constantes

Para que um programa possa manipular qualquer dado, é necessário identificar explicitamente onde o mesmo será armazenado para que se possa recuperá-lo e realizar as operações e cálculos necessários. Esses dados podem ser manipulados através de variáveis e constantes. Para tanto, variáveis e constantes devem ser devidamente criadas. O ato de criar uma variável ou constante é denominado “declaração”: para ambos é necessário estabelecer o seu nome (identificador); e também o tipo no caso da variável, e o valor no caso da constante.

2.2.1. Declaração de Constantes

Como foi visto anteriormente, sempre que um determinado valor utilizado em um algoritmo permanece inalterado, dizemos que se trata de uma constante. Este valor constante pode ser escrito diretamente no algoritmo ou pode ser utilizado através de identificadores, ou seja, da mesma forma que as variáveis, as constantes também podem ser declaradas, no entanto, não é necessário especificar o seu tipo e seu valor deve ser designado explicitamente, conseqüentemente, este valor é que determina o tipo da constante.

Declaração de constantes - Sintaxe

constante <identificador> = <valor>;

onde <identificador> é apenas um identificador e <valor> é o valor constante atribuído à constante; o identificador deve ser separado do seu valor pelo sinal de igual (“=”) e logo após deve haver o sinal de ponto e vírgula (“;”).

Nota1: a palavra reservada¹ **constante** é utilizada uma única vez para indicar que se inicia a seção de declaração de constantes; havendo necessidade de declarar mais constantes, basta somente especificar o seu identificador com seu respectivo valor logo a seguir.

Nota2: valores constantes do tipo caracter devem ser delimitados por aspas duplas (“”).

Nota3: constantes do tipo lógico assumem valor *verdadeiro* ou *falso*.

Exemplos: declaração de uma constante com o valor de π e outra com a frase “Proibido Fumar!”.

```
constante PI = 3.1416;
```

```
Mensagem= "Proibido Fumar !";
```

2.2.2. Declaração de variáveis

Sempre que for necessário armazenar algum valor a ser utilizado pelo algoritmo para realizar uma determinada operação, e não for possível determinar precisamente qual é este valor, uma variável deve ser devidamente criada. O ato de criar uma variável é conhecido como declaração de variável. Para cada variável a ser criada, deve-se estabelecer explicitamente o seu nome e o seu tipo (que pode ser um dos tipos primitivos

visto anteriormente, conseqüentemente o valor que se deseja armazenar na variável vai determinar o seu tipo). Por exemplo, se desejamos armazenar a nota bimestral de um aluno podemos lhe dar o nome de *NotaBim* e este deve ser do tipo *real*; se desejamos armazenar o nome de um produto, podemos lhe dar o nome de *NomeProd* este deve ser do tipo *caracter*, se desejamos armazenar a idade de uma pessoa podemos lhe dar o nome de *idade* e este deve ser do tipo *inteiro*. Sempre que for necessário manipular algum dado, deve-se fazer menção ao nome da variável na qual o dado em questão está (ou será) armazenado.

Declaração de variáveis - Sintaxe
<i>variável</i> <lista de identificadores>: <tipo>;
onde <lista de identificadores>, pode ser apenas um identificador ou vários separados por vírgula e <tipo> pode ser um dos tipos primitivos (inteiro, real, caracter ou lógico); a lista de identificadores deve ser separada do tipo por dois pontos (":") e após o tipo deve haver o sinal de ponto e vírgula (";").

Nota: de forma semelhante a declaração de constante, a *palavra reservada* *variável* é utilizada uma única vez para indicar que se inicia a seção de declaração de variáveis; havendo necessidade de declarar mais variáveis, basta apenas indicar seus identificadores com seus respectivos tipos.

Exemplos: declaração de variáveis para armazenar o raio e a área de uma circunferência, e a idade e altura de uma pessoa.

```
variável raio, area, altura: real;
idade: inteiro;
```

3. Expressões Aritméticas (operadores aritméticos / funções matemáticas)

As expressões aritméticas devem ser escritas linearmente usando uma notação matemática própria da linguagem algorítmica. Esta notação estabelece as seguintes regras:

- Mantêm-se os símbolos "+" e "-" para as operações de adição e subtração.
- Para multiplicação e divisão, utilizam-se os símbolos "*" e "/" respectivamente.
- São utilizados dois novos operadores para divisão de números inteiros *div* (tem-se como resultado a parte inteira do quociente) e *resto* (tem-se como resultado o resto da divisão).

Prioridade entre os operadores aritméticos	
1ª	* / div resto
2ª	+ -

¹ Palavra reservada é um termo que designa os elementos terminais da gramática.



Nota: quando a prioridade for a mesma dentro de uma expressão, devemos resolvê-la da esquerda para direita. Se quisermos forçar a mudança da prioridade, devemos colocar entre parênteses a expressão que deve ser resolvida primeira.

Exemplos: $(5 + 7) * 4 + 10 / 2$ (1ª operação – 5+7: precedência forçada através dos parênteses)

$12 * 4 + 10 / 2$ (2ª operação – $12*4$)

$48 + 10 / 2$ (3ª operação – $10/2$)

$48 + 5$ (4ª e última operação – $48+5$)

 53 (resultado da expressão)

 10 *div* 3 (resulta em 3 – parte inteira do quociente de 10 por 3)

 10 *resto* 3 (resulta em 1 – resto da divisão de 10 por 3)

3.1. Funções Matemáticas

Além das operações básicas citadas, podemos usar algumas funções matemáticas. Dentre elas:

função	Resultado da função	tipo do resultado	exemplo
abs (exp)	Valor absoluto da expressão (elimina o sinal)	real ou inteiro	abs(-10) = 10
int (exp)	Valor inteiro da expressão (ignora a parte decimal)	inteiro	int (4.25) = 4
quad (exp)	Quadrado da expressão (eleva ao quadrado)	real ou inteiro	quad (3) = 9
rq (exp)	raiz quadrada da expressão	real	rq (36) = 6.0

Exemplo: considerando as variáveis A, B, C e D com os respectivos valores 3, 5.5, -10 e 8

$D \text{ resto } A + 6 * A - \text{int} (B) + \text{quad} (\text{abs}(D))$

$2 \quad + \quad 18 \quad - \quad 5 \quad + \quad \text{quad} (10)$

$15 \quad + \quad 100$

115

4. Expressões Lógicas

Expressões Lógicas são expressões que apresentam operadores lógicos e/ou operadores relacionais e cujos operandos são relações e variáveis/constantes do tipo lógico, ou seja, Verdadeiro ou Falso.

4.1. Relações

Uma relação é composta por operadores relacionais e dois valores do mesmo tipo. Basicamente, é feita uma comparação entre estes dois valores do mesmo tipo, obtendo-se como resultado um valor lógico

(Verdadeiro – V ou Falso – F). Essa comparação pode ser entre dois operandos (expressões) e um operador relacional (veja a tabela a seguir).

Operadores Relacionais			
Símbolo	Descrição	Exemplo	Resultado
=	Igual	3=3	V
>	maior	2+5 > 9	F
<	menor	2+1 < 5	V
< >	Diferente	"noite" < > "dia"	V
<=	menor ou igual	2<=10	V
>=	maior ou igual	2>=10	F

4.2. Operadores Lógicos

Os operadores lógicos são utilizados para combinarem relações que têm como resultado um valor lógico. Basicamente, podemos fazer uso de três operadores lógicos: **e**, **ou** e **não**. Supondo A e B relações que têm como resultado Verdadeiro ou Falso, a combinação entre eles utilizando os operadores lógicos se dá conforme as tabelas-verdades a seguir.

A	B	A e B
F	F	F
F	V	F
V	F	F
V	V	V

A	B	A ou B
F	F	F
F	V	V
V	F	V
V	V	V

A	não A
F	V
V	F

Para deixar mais claro a utilização dos operadores lógicos, suponha as sentenças a seguir:

A= "a aula é no período da tarde"

B= "está calor"

Uso do operador lógico **e**:

"Se a aula for no período da tarde **e** estiver calor, Pedro vai à escola."

Pedro só vai à escola se A e B ocorrerem (forem verdadeiros) simultaneamente. A utilização do operador lógico **e** impõe que todas as relações combinadas na expressão lógica sejam verdadeiras para que o resultado da expressão seja verdadeiro.

"Se a aula for no período da tarde **ou** estiver calor Pedro vai à escola."

Pedro vai a escola se A ou B (um dos dois) ocorrer (for verdadeiro). O operador lógico **ou** determina que se qualquer uma das relações combinadas na expressão lógica for verdadeira o resultado da expressão é verdadeiro.

“Se **não** estiver calor, Pedro vai à escola.”

Pedro vai à escola se B não ocorrer (for falso). Dessa forma a expressão **não** “estiver calor” será verdadeira.

5. Comando de Atribuição

Um comando de atribuição estabelece que uma determinada variável deve assumir um determinado valor. Para tanto, é utilizado o operador de atribuição (<-); o que for resolvido do lado direito do operador é atribuído à variável do lado esquerdo.

Comando de atribuição - Sintaxe
<identificador> <- <expressão>;
onde <identificador> é o nome de uma variável na qual se deseja armazenar algo, e <expressão> pode ser qualquer símbolo ou combinações de símbolos que resulte em um determinado valor.

Nota: o valor resultante da resolução da expressão deve ser do mesmo tipo da variável. Não seria possível, por exemplo, atribuir um valor do tipo caracter a uma variável do tipo inteiro.

Exemplo: \suponha a declaração da constante e das variáveis a seguir

constante PI = 3.1416;
variáveis raio, area: real;
\veja as atribuições
raio<-5.0; \o valor 5.0 é atribuído a variável raio
area<-PI*raio*raio; \uma expressão aritmética é atribuída à variável area, ou seja,
 \ a expressão do lado esquerdo é resolvida e seu resultado é
 \ atribuído à variável

6. Comandos de Entrada / Saída

Para qualquer algoritmo, seria desejável que dados do mundo real pudessem ser introduzidos como entrada e os resultados processados pelo algoritmo também pudessem ser exibidos ao mundo real. Para permitir tais situações, pode-se contar com os comandos de entrada e saída. Através de um comando de entrada, o algoritmo pode obter dados do usuário para efetuar determinados processamentos; uma vez processadas as informações são exibidas ao usuário através de um comando de saída.

6.1. Entrada de Dados

Comando de leitura - Sintaxe
leia (<lista de variáveis>;
onde <lista de variáveis> pode ser o identificador de uma ou mais variáveis; sendo mais de uma, estas devem ser separadas por vírgula.

Nota: durante o processo de entrada, os dados fornecidos como entrada são “atribuídos” às variáveis identificadas na lista de variáveis.

Exemplos:

```
leia (raio); \ao encontrar um comando de leitura, o programa é suspenso
              \esperando uma entrada do usuário
leia (idade, altura); \neste exemplo, o primeiro dado fornecido como entrada
              \será armazenado na primeira variável e,
              \consequentemente, o segundo dado na segunda variável
```

6.2. Saída de Dados

Comando de escrita - Sintaxe
<i>escreva</i> (<lista de dados>);
onde <lista de dados> pode ser uma constante alfanumérica ou numérica, um identificador de variável ou constante ou uma expressão; mais de um elemento de dado pode ser escrito e, para tanto, cada elemento deve ser separado por vírgula.

Nota1: ao especificar um identificador de constante ou variável, o seu valor armazenado é exibido.

Nota2: uma expressão aritmética também pode ser incluída na lista de dados; neste caso a expressão é resolvida e é exibido o seu resultado final.

Exemplos:

```
escreva (area); \exibe o conteúdo da variável área através de um dispositivo de
              \saída
escreva ("Qual a sua idade?"); \exibe a constante alfanumérica
escreva ("Você tem ", idade, "anos e pesa ", peso, "quilos."); \exibe a constante
              \caracter "Você tem", o conteúdo da variável idade, a constante caracter
              \ "anos e pesa", o conteúdo da variável peso e a constante "quilos."
```

7. Estrutura de um algoritmo

Um programa em algoritmo deve obedecer a uma sintaxe predefinida pela sua gramática formal. Para a linguagem algorítmica em questão, um programa em algoritmo deve começar com a palavra reservada *algoritmo* seguida de um identificador que seria o nome do programa. A seguir, teríamos a declaração das constantes e das variáveis, necessariamente nesta ordem. Após estas declarações, começaria a parte executável do algoritmo composto por instruções, denominado bloco de comando.

Um bloco de comando pode ser definido como um conjunto de ações (comandos em algoritmo) com funções devidamente definidas para atingir um determinado objetivo. Um bloco é delimitado pelas palavras reservadas *inicio* e *finalgoritmo*. Internamente a estes delimitadores podem ser estabelecidos comandos (qualquer comando visto até então – atribuição, leitura e escrita - e os demais a serem vistos ainda), que indicam as ações que o algoritmo deverá realizar.

Estrutura de um Algoritmo - Sintaxe

Algoritmo <identificador do algoritmo>

<declaração de constantes>

<declaração de variáveis>

início

<bloco de comandos>

finalgoritmo.

onde <identificador do algoritmo> é o nome dado ao algoritmo, <declaração de constantes> é zona onde são declaradas todas as constantes do algoritmo, <declaração de variáveis> é a zona onde são declaradas todas as variáveis do algoritmo e <bloco de comandos> são todas as ações que devem ser realizadas pelo algoritmo (pode ser incluído qualquer comando visto até então (atribuição, leitura e escrita) e demais comandos que ainda serão vistos).

Exemplo: Programa em algoritmo que calcula a área de uma circunferência

Algoritmo Area_Circunferencia**constante** PI=3.1415;**variavel** raio, area: real;**início** **escreva** ("Forneça o valor do raio da circunferência: "); **leia** (raio); Area <- PI***quad** (raio); **escreva** ("O valor da área é ", area);**finalgoritmo.**

8. Estruturas de Seleção

Caso não seja estabelecido o contrário, um algoritmo é composto por uma "seqüência" de comandos que devem ser executados seqüencialmente (uma linha após a outra). No entanto, em algumas situações é desejável que certos comandos sejam executados em determinadas circunstâncias. Neste caso, pode-se estabelecer uma condição para que estes sejam executados. Suponha, por exemplo, que seu algoritmo tenha que exibir a mensagem "Produto caro!" somente se o preço de um produto fornecido pelo usuário for superior a 20 reais. Para poder tratar esta e as demais situações similares, pode-se contar com estruturas de seleção que permitem a escolha de uma seqüência de comandos para ser executada quando determinadas condições (representadas por expressões lógicas) são ou não satisfeitas.

Comando SE – Sintaxe		
Estrutura Condicional Simples	Estrutura Condicional Composta	Estrutura Condicional Aninhada
<i>Se</i> (<condição>) <i>Entao</i> <comando> <i>Fimse</i> ;	<i>Se</i> (<condição>) <i>entao</i> <comando1> <i>senao</i> <comando2> <i>fimse</i> ;	<i>Se</i> (<condição1>) <i>entao se</i> <condição2> <i>entao</i> <comando1> <i>senao</i> <comando2> <i>fimse</i> ; <i>senao</i> <comando3> <i>fimse</i> ;

onde <condição>, <condição1> e <condição2> são expressões lógicas, ou seja, seu resultado é sempre Verdadeiro ou Falso, <comando>, <comando1>, <comando2> e <comando3> podem designar qualquer comando ou sequência de comandos da linguagem algorítmica.

- ✓ Condicional Simples: somente se a expressão em <condição> tiver resultado *Verdadeiro* o(s) comando(s) especificados a partir da cláusula *entao* serão executados.
- ✓ Condicional Composta: semelhante ao anterior, com a inclusão da cláusula *senao* - se a expressão em <condição> for *Falsa* o(s) comando(s) especificados a partir da cláusula em questão serão executados.
- ✓ Condicional Aninhada: como foi mencionado, nas cláusulas *entao* e *senao* pode ser especificados quaisquer comandos, inclusive o próprio comando *se*.

Nota: a condição (expressão lógica) deve ser delimitada por parênteses.

Exemplo 1: Programa em algoritmo que verifica se as medidas de três lados formam um triângulo, em caso afirmativo exibe a mensagem “As medidas são valores válidos para formar um triângulo”, caso contrário “As medidas não formam um triângulo”.

Algoritmo Triangulo

variável A, B, C: real;

início

```
escreva ('Forneça três medidas para os lados de um triângulo: ');  
leia (A,B, C);  
se ((A < B+C) e (B < A+C) e (C < A+B) )  
    entao escreva ("As medidas são valores válidos para formar um triângulo");  
    senao escreva ("As medidas não formam um triângulo");  
fimse;
```

finalgoritmo.

Exemplo 2: Programa em algoritmo que verifica se as medidas de três lados formam um triângulo e, em caso afirmativo, verifica se se trata de um triângulo equilátero (três lados de comprimentos iguais), isósceles (dois lados quaisquer de comprimentos iguais) ou escaleno.

Algoritmo Triangulo

variável A, B, C: real;

início

escreva ("Forneça três medidas para os lados de um triângulo: ");

leia (A, B, C);

se ((A < B+C) e (B < A+C) e (C < A+B))

entao escreva ("As medidas são valores válidos para formar um triângulo");

se ((A=B) e (B=C))

entao escreva ("Formam um triângulo equilátero");

senao se ((A=B) ou (B=C) ou (A=C))

entao escreva ("Formam um triângulo isósceles");

senao escreva ("Formam um triângulo escaleno");

fimse;

fimse;

senao escreva ("As medidas não formam um triângulo");

fimse;

finalgoritmo.

9. Estruturas de Repetição

Freqüentemente, em um "programa" precisamos efetuar uma mesma seqüência de ações por várias vezes. Para determinar tal situação, podemos estabelecer um "laço" utilizando uma estrutura de repetição. Na linguagem algorítmica em questão, podemos fazer uso de três estruturas de repetição que podem ser aplicadas distintamente, dependendo da situação que se deseja tratar.

Comando Repita (teste no fim do laço) - Sintaxe

Repita

<comando>

ate (<condição>)

Onde <condição> é uma expressão lógica, ou seja, seu resultado é sempre **Verdadeiro** ou **Falso**; <comando> pode ser qualquer comando ou seqüência de comandos da linguagem algorítmica. Este(s) comando(s) são delimitados pela palavras reservadas **repita** e **ate** e executados até que a expressão em <condição> seja verdadeira.

Nota1: como o teste é feito no fim do laço, a execução dos comandos especificados dentro do laço em questão será feita no mínimo uma vez.

Nota2: a condição (expressão lógica) deve ser delimitada por parênteses.

Exemplo:

repita

escreva ("Oi!"); \a frase Oi! será escrita até que o usuário digite o valor ZERO

leia (numero);

ate (numero=0);

Comando Enquanto (teste no início do laço) – Sintaxe
<i>Enquanto</i> (<condição>) <i>faça</i> <comando> <i>fimenquanto</i> ;
onde <condição> é uma expressão lógica, ou seja, seu resultado é sempre Verdadeiro ou Falso, <comando> pode ser qualquer comando ou sequência de comandos da linguagem algorítmica. Este(s) comando(s) são delimitados pela palavras reservadas <i>enquanto</i> e <i>fimenquanto</i> e executados enquanto a expressão em <condição> for Verdadeira.

Nota: como o teste é feito no início do laço, é possível que a execução dos comandos especificados dentro do laço em questão não ocorra uma vez sequer, caso a condição já seja FALSA, inicialmente.

Exemplo:

```
i<- 1;
enquanto (i <=10) faça
    escreva ('Oi !'); \a frase 'Oi !' será escrita 10 vezes
    i<-i+1;
fimenquanto;
```

Comando Para - Sintaxe
<i>Para</i> <variável de controle> <i>de</i> <valor inicial> <i>ate</i> <valor final> <i>incremento</i> <valor incremento> <i>faça</i> <comando> <i>fimpara</i> ;
onde <variável de controle> é uma variável do tipo inteiro, <valor inicial>, <valor final> e <valor incremento> são constantes numéricas ou expressões aritméticas do tipo inteiro e <comando> pode ser qualquer comando ou sequência de comandos da linguagem algorítmica. A variável de controle determina o número de vezes que este(s) comando(s) será (ao) executado(s), partindo do valor inicial e sendo incrementado o valor especificado na cláusula <i>incremento</i> até atingir o valor final (inclusive o valor final). A utilização da cláusula <i>incremento</i> é opcional e indica o valor do incremento da variável de controle; quando omitida assume-se que o incremento será de uma unidade.

Nota: É recomendável o uso do laço *para* quando se souber precisamente o número de iterações que se deve realizar.

Exemplo 1:

```
para i de 1 até 10 faça
    escreva ('Oi !'); \a frase 'Oi !' será escrita 10 vezes
fimpara;
```


Exemplo 2:

```
para i de 0 a 100 incremento 2 faca \o primeiro valor de i é 0 e com incremento
                                     \ igual a 2 será 2, 4, 6 e assim sucessivamente
    escreva (i); \serão escritos os valores 0, 2, 4, 6, ...100, ou seja todos os
                \números pares de 0 a 100
fimpara;
```

10. Variáveis Compostas (tipos definidos pelo programador)

Na seção 3.2.2 vimos que podemos declarar variáveis como sendo dos tipos inteiro, real, carácter ou lógico (tipos primitivos). O tipo determina uma classificação entre as variáveis e, conseqüentemente, estabelece o domínio de valores aos quais elas estão restritas.

À medida que os algoritmos vão se tornando mais complexos, é possível que a manipulação de dados através de variáveis de tipos primitivos não seja satisfatória, sendo necessária a existência de tipos mais complexos que permitam agrupar e organizar dados. Dessa forma, seria interessante que o programador pudesse ir criando novos tipos de acordo com sua necessidade, fazendo uso da composição de tipos primitivos. Denominamos estes novos tipos de *estrutura de dados*. Uma estrutura de dado determina como os tipos primitivos estão organizados. Inicialmente, podemos definir dois tipos de estrutura de dados: vetor e registro.

10.1. Vetor - Variável Composta Homogênea

Um vetor é considerado uma estrutura do tipo Variável Composta Homogênea, por tratar-se de um agregado de elementos do mesmo tipo. Cada elemento ocupa uma determinada posição na estrutura de dados e pode ser referenciado por um número que indica esta posição. Este número é denominado índice e dizemos que o vetor é uma estrutura indexada. Se o vetor possui apenas um índice, dizemos que o mesmo é unidimensional; bidimensional e tridimensional para dois e três índices. Veja os exemplos a seguir

47	12	19	31	4
1	2	3	4	5

Exemplo: Vetor unidimensional, tendo como elemento um valor do tipo inteiro (apresenta um índice cujo domínio abrange valores de 1 a 5, conseqüentemente a estrutura apresenta 5 posições). Para referenciar um elemento, basta indicar a sua posição (posição 3 = 19).

1	2	1	15	42	19
2	11	23	29	35	14
3	4	7	16	25	40
	1	2	3	4	5

Exemplo: Vetor bidimensional, tendo como elemento um valor do tipo inteiro (apresenta dois índices: de 1 a 3 e de 1 a 5 que, respectivamente, indicam o número da linha e da coluna onde se encontra um determinado elemento). Para referenciar um elemento, devemos indicar a linha e a coluna onde ele se encontra (linha 2, coluna 3 = 29).

1	2	1	15	42	19
2	11	23	29	35	14
3	4	7	16	25	40
	1	2	3	4	5

plano 1

1	2	1	15	42	19
2	11	23	29	35	14
3	4	7	16	25	40
	1	2	3	4	5

plano 2

Exemplo: Vetor tridimensional, tendo como elemento um valor do tipo inteiro (apresenta três índices: de 1 a 2, de 1 a 3 e de 1 a 5 que, respectivamente, indicam os números do plano, da linha e da coluna onde se encontra um determinado elemento). Para referenciar um elemento, devemos indicar o plano, a linha e a coluna onde ele se encontra (plano 2, linha 3, coluna 5 = 40).

Definição de um tipo vetor – Sintaxe

tipo <identificador do tipo vetor> = *vetor* [<definição dos índices>] *de* <tipo>;

onde <nome do tipo vetor> é o nome do tipo sendo criado (este nome é acrescentado à lista de tipos primitivos o que permite a declaração de uma variável do novo tipo em questão); <definição dos índices> determinam quantas dimensões o vetor vai possuir (no máximo 3) e quantos elementos haverá em cada dimensão, e <tipo> é o tipo estabelecido para cada componente do vetor (pode ser um tipo primitivo – inteiro, real, caracter ou lógico) ou um tipo definido anteriormente pelo programador.

Nota1: para cada dimensão, deve ser estabelecido um valor. Por exemplo, se o vetor possuir apenas uma dimensão e este valor for 5, então terá 5 elementos; se o vetor possuir duas dimensões com valores 3 (linhas) e 5 (colunas) então terá 15 elementos (3x5); se o vetor possuir três dimensões com valores 3(planos), 3 (linhas) e 4 (colunas) então terá 36 elementos (3x3x4).

Nota2: se possuir mais de uma dimensão, seus valores devem vir precedidos por vírgula. Por exemplo, duas dimensões: 3, 4 (3 linhas e 4 colunas).

Nota3: uma vez definido um tipo, variáveis podem ser declaradas como sendo deste tipo; a sintaxe para a declaração de variáveis é a mesma dos tipos primitivos (<identificador do tipo> : <lista de variáveis>;)

Exemplo:

```
tipo tabela=vetor [10] de inteiro; \definição de um tipo vetor chamado tabela com
                                     \10 elementos
variavel T: tabela; \declaração de uma variável com sendo de um tipo definido
                                     \pelo programador.
```


Acesso a variável do tipo vetor – Sintaxe

<identificador da variável do tipo vetor> [/<expressão>]

onde <identificador da variável do tipo vetor> é o nome da variável declarada como tipo vetor, e <expressão> pode ser uma variável do tipo inteiro, uma constante numérica inteira ou uma expressão aritmética que resultará num número inteiro.

Nota: para acessar um componente do vetor, devemos especificar a sua posição utilizando um valor inteiro que deve ser maior ou igual a 1, e menor ou igual a <qtde de elementos>.

Exemplo: considerando as declarações do exemplo anterior:

```
T[4]<- 13;  \o valor 13 será armazenado na posição 4 do vetor T
leia (T[2]); \o valor lido será armazenado na posição 2 do vetor T
para I de 1 até 5 faça \laço para entrada de valores no vetor
    escreva ('Forneça um número inteiro:');
    leia (T[I]); \leitura de cada componente do vetor através do laço
fimpara;
```

10.2. Registro - Variável composta heterogênea

Um registro é considerado uma estrutura do tipo Variável Composta Heterogênea por tratar-se de um agregado de elementos de tipos diferentes. Diferentemente do vetor, onde os elementos são indexados (ou seja, referencia-se a posição do elemento que se deseja acessar), cada elemento de um registro está associado a um nome, denominado nome do campo, e o acesso é feito referenciando-se este nome.

Exemplo:

Código	1287843
Nome	Asdrubal Marcondes
Estado Civil	Casado
Salário	1.027,00

Definição de um tipo registro - Sintaxe

tipo < identificador do tipo registro>: *registro*

<campo1>: <tipo1>;
<campo1>: <tipo2>;
...
<campoN>: <tipoN>;
fimregistro;

onde < identificador do tipo registro> é o nome do tipo sendo criado; <campo1> .. <campoN> são os componentes do registro, cada qual com seu respectivo tipo (<tipo1> .. <tipoN>), que pode ser um tipo primitivo – inteiro, real, caracter ou lógico - ou um tipo definido pelo programador (veja logo a seguir).

Nota: quando mais de um campo forem do mesmo tipo, pode-se estabelecer uma lista de identificadores separados por vírgula, seguidos de uma única definição de tipo.

Exemplo:

```
tipo funcionario=registro \definição de um tipo registro chamado funcionario
    codigo, nome, est_civil: caracter;
    salario: real;
    fimregistro;
variavel F:funcionario; \declaração de uma variável do tipo funcionario
```

Acesso a variável do tipo registro – Sintaxe

<identificador da variável do tipo registro>.<nome do campo>

onde <identificador da variável do tipo registro> é o nome da variável declarada como um tipo registro, e <nome do campo> é o nome de um dos componentes do registro.

Exemplo:

leia (F.nome); \acesso ao campo *nome* do registro *F*

10.3. Variáveis Compostas Complexas

Com a definição de novos tipos, podemos criar estruturas complexas combinando vetores com registros e vice-versa.

Pode ser necessário armazenar o código de funcionário, o seu nome, estado civil e valor do salário. Suponhamos que o armazenamento destes dados se estenda para uma empresa com vários funcionários, sendo desejável que os dados referentes a um determinado funcionário fossem agrupados numa estrutura. Situações como esta exigem a especificação de estruturas mais complexas.

Exemplo:

```
tipo funcionario=registro \definição de um tipo registro chamado funcionario
    codigo, nome, est_civil: caracter;
    salario: real;
    fimregistro;
empresa=vetor [100] de funcionario; \definição de um tipo vetor chamado
    \empresa sendo que cada componente deste vetor é do tipo funcionario.
variavel fabrica:empresa; \declaração da variável fabrica como sendo do
    \tipo empresa.
```

Para acessar uma variável como a definida no exemplo, devemos partir da estrutura mais externa para a interna. A variável **fabrica** é do tipo **empresa** que é um vetor, neste caso devemos referenciar a posição do vetor que desejamos acessar.

fabrica[<posição desejada>]

Como cada posição do vetor é do tipo funcionário, que é um registro, devemos especificar qual campo desejamos acessar

fabrica[<posição desejada>].<campo desejado>

Sendo assim:

Leia (fabrica[53].nome); \leitura do campo nome do registro localizado na posição
 \53 do vetor

Uma outra situação seria o armazenamento da ficha de um aluno com seu número de matrícula, nome, curso em que está matriculado, número de faltas referentes aos meses do 1.o semestre e as duas notas bimestrais de cada disciplina (total de 6).

Partimos da estrutura mais interna: para armazenar faltas e notas, definimos duas estruturas:

```
Tipo falta=vetor [5] de inteiro; \número de faltas referente a 5 meses
nota= vetor[2] de real; \duas notas bimestrais
disciplina=registro
    nome:caracter; \nome da disciplina
    f:falta; \numero de faltas ocorridas nos 6 meses
    n:nota; \2 notas obtidas nas avaliações
    fimregistro;
semestre=vetor [6] de disciplina; \dados referentes a 6 disciplinas
aluno=registro
    n_matricula:caracter; \numero da matricula
    nome, curso:caracter; \nome do aluno e nome do curso
    sem:semestre; \dados referentes a notas e faltas das
        \ disciplinas que está cursando
    fimregistro;

variavel a:aluno;
```

11. Módulos de Algoritmos (Sub-rotinas)

Todos os algoritmos vistos até o momento apresentam uma estrutura monolítica, ou seja, existe um único bloco onde se inicia e termina a sua execução, sendo as instruções executadas sequencialmente, salvo em algumas situações em que são utilizados os comandos de seleção. No entanto, seria desejável que um algoritmo complexo pudesse ser dividido em vários blocos independentes para poder tratar cada situação individualmente, em vez de lidar com o problema como um todo. Esses blocos independentes são denominados sub-rotinas.

A principal razão para se criar sub-rotinas está relacionada à otimização do código do algoritmo (quando um determinado trecho de comandos deve ser repetido por várias vezes, pode-se colocá-lo em uma

sub-rotina e referenciá-la quantas vezes se desejar) e, também à estruturação (organização) do algoritmo (pode-se dividir o algoritmo em várias sub-rotinas, cada uma com uma função bem definida e implementá-la parte por parte).

Uma sub-rotina deve ser inicialmente criada para seu uso posterior. Da mesma forma que variáveis são declaradas para armazenar valores, sub-rotinas devem ser declaradas para poderem ser executadas.

A declaração de sub-rotinas deve ser feita após a declaração das variáveis globais e antes do início do bloco principal do algoritmo do qual o mesmo fará parte. A sub-rotina deve agrupar uma sequência bem definida de comandos responsáveis pela realização de uma determinada tarefa.

Algoritmo <identificador do algoritmo>
<declaração de constantes>
<definição de tipos>
<declaração de variáveis>
<declaração de sub-rotinas>
início
<bloco de comandos>
finalgoritmo.

Na execução de uma sub-rotina, o algoritmo principal (da qual a sub-rotina faz parte) evoca a sua chamada identificando-o pelo nome e lhe transfere o controle de execução; uma vez finda a sub-rotina, o controle de execução retorna ao algoritmo principal.

Neste trabalho serão vistas duas formas básicas de sub-rotinas, cada uma com sua peculiaridade e finalidade específica:

- Procedimentos
- Funções

A diferença básica entre os procedimentos e as funções é que as funções obrigatoriamente retornam um valor ao término de sua execução e os procedimentos não. Fazendo uma analogia com algumas instruções da linguagem algorítmica vista até o momento, pode-se dizer que uma sub-rotina do tipo função é semelhante às funções matemáticas pré-definidas como o *int*, *abs*, *rq* ou *quad*, a sua execução resulta num determinado valor e o identificador da função deve ser referenciado numa expressão; e uma sub-rotina do tipo procedimento pode ser vista como um comando *leia* ou *escreva*.

11.1. Declaração e Chamada de Procedimento

Declaração de procedimento - Sintaxe
<pre> procedimento < identificador do procedimento>; <declarações de tipos, constantes, variáveis e sub-rotinas locais> inicio <comandos pertencentes à subrotina> fimprocedimento;</pre>
onde <identificador o procedimento> é o nome dado ao procedimento; <declaração de tipos, constantes, variáveis e sub-rotinas locais> são os tipos, as constantes, as variáveis e também as sub-rotinas declaradas localmente, e <comandos pertencentes à subrotina> são os comandos que devem ser executados quando o procedimento for chamado.

Nota: os tipos, as constantes, as variáveis e também as sub-rotinas declaradas localmente só poderão ser referenciados dentro dos comandos pertencentes à sub-rotina (veja maiores detalhes em problema de escopo).

Exemplo: fazer um algoritmo que leia 40 registros contendo o RA, o nome e a média do aluno. Exibir o RA e nome de cada aluno e informar se o mesmo está aprovado (media ≥ 6.0) ou reprovado. (declarar um procedimento para a entrada de dados e um outro para verificar a aprovação).

```

Algoritmo
  Constante N=40;
  Tipo Aluno = registro
      Caracter: RA, Nome;
      Real: Media;
      Fimregistro;
  Tipo Turma=vetor [1..N] de Aluno;
  Variavel A: Turma;
  Procedimento CadastraMedia;
  Variavel I:inteiro;
  Início
      Para I de 1 até N faça
          Escreva ('RA do Aluno: ');
          Leia (A[I].RA);
          Escreva ('Nome do Aluno: ');
          Leia (A[I].Nome);
          Escreva ('Média do Aluno: ');
          Leia (A[I].Media);
```

```

Fimpara;
Fim;
Procedimento InformaAprovado;
Variável I: inteiro;
Início
    Para I de 1 até N faça
        Escreva ('RA: ', A[I].RA);
        Escreva ('Nome: ', A[I].Nome);
        Se A[I].Media  $\geq 6.0$ 
            Então Escreva ('Aprovado');
            Senão Escreva ('Reprovado');
        Fimse;
    Fimpara;
Fim;
Início
    CadastraMedia;
    InformaAprovado;
Fim.
```

Talvez no exemplo dado não fique caracterizada a otimização de código, uma vez que os comandos das subrotinas são executados apenas uma vez. No entanto, suponha que deseje cadastrar a média para verificar se o aluno foi aprovado ou não em diversas disciplinas. Nesta situação, o corpo principal do algoritmo ficaria da seguinte forma:

....

Início

```
escreva ('Forneça as médias de Lógica de Programação');
CadastraMedia;
Escreva ('Situação dos alunos de Lógica de Programação');
InformaAprovado;
Escreva ('Forneça as médias de Geometria Analítica');
CadastraMedia;
Escreva ('Situação dos alunos de Lógica de Programação');
InformaAprovado;
Escreva ('Forneça as médias de Introdução à Ciência da Computação');
CadastraMedia;
Escreva ('Situação dos alunos de Introdução à Ciência da Computação');
InformaAprovado;
```

Fim algoritmo.

11.2. Declaração e Chamada de Funções

A característica fundamental de uma função é que ela retorna um valor para o programa que a chamou. Assim, a sua declaração deverá conter o tipo de dado que irá retornar.

Declaração de Função - Sintaxe
<pre><i>funcao</i> <identificador da função> : <tipo da função>; <declarações de tipos, constantes, variáveis e sub-rotinas locais à função> <i>inicio</i> <comandos pertencentes à função> <i>retorna</i>(<valor de retorno>); <i>fimfuncao</i>;</pre>
onde <identificador da função> é o nome dado à função; <tipo da função> é o tipo do valor que a função em questão deve retornar; <declaração de tipos, constantes, variáveis e sub-rotinas locais> são os tipos, as constantes, as variáveis e também as sub-rotinas declaradas localmente, ou seja, que só poderão ser referenciados dentro dos <comandos pertinentes à função>, e <valor de retorno> é o valor resolvido dentro da função que deve ser retornado.

Nota: dentro de uma função, a última linha de comando a ser executada deve ser uma atribuição envolvendo o identificador da função e o valor que a função em questão deve retornar.

Exemplo: algoritmo para calcular o maior entre 3 valores lidos pelo teclado.

Algoritmo MostraMaior;

Variavel A, B, C, M: inteiro;

Funcao ComparaMaior: inteiro

Variavel maior: inteiro;

Início

Se (A>B) e (A>C)

Entao maior<-A;

Senao se (B>C)

Entao maior<-B;


```
        Senao maior<-C;
    Fimse;
Fimse;
    ComparaMaior<-maior; \O nome da função recebe o valor a ser retornado
Fim;
Início {Programa Principal}
    Escreva ('Forneça 3 números inteiros: ');
    Leia (A, B, C);
    M<-ComparaMaior;
    Escreva (' O maior número é', M);
fim.
```

11.3. Parâmetros de Sub-rotina

Dentro de uma sub-rotina (procedimento ou função), embora possamos acessar as variáveis globais, isto não se constitui numa boa prática de programação. A transferência de informações para as sub-rotinas podem ser formalizadas e documentadas através de parâmetros. Denominamos “parâmetros de definição” os parâmetros declarados no cabeçalho da sub-rotina e “parâmetros de chamada” os argumentos a serem associados aos parâmetros de definição na chamada da sub-rotina.

A utilização de parâmetros para formalizar a comunicação entre sub-rotinas, além de servir como uma documentação do programa elaborado, permite que uma mesma sub-rotina seja ativada com operandos (valores de parâmetros) diferentes, o que possibilita ter resultados de processamento diferentes, para diferentes grupos de dados de entrada.

Podemos encontrar um exemplo prático em alguns comandos da linguagem algorítmica: suponha o comando “escreva” como sendo um procedimento; na sua chamada devemos passar os argumentos de escrita como parâmetros de entrada, a cada chamada do comando a execução se dará de forma diferente dependendo do valor argumento que é passado como parâmetro.

O uso de parâmetros permite tanto fornecer dados de entrada às sub-rotinas como receber saídas em decorrência do seu processamento. Quando o objetivo é somente fornecer dados de entrada às sub-rotinas diz-se que os parâmetros são passados por valor, ao passo que, quando se desejar que a execução da sub-rotina gere saídas os parâmetros são passados por referência.



Declaração de Parâmetros - Sintaxe

procedimento < identificador do procedimento> (<tipo de passagem> <identificador do parâmetro>: <tipo>, <tipo de passagem> <identificador do parâmetro>: <tipo>...);

funcao < identificador da função> (<tipo de passagem> <identificador do parâmetro>: <tipo>; ...): <tipo da função>;

onde <tipo de passagem> determina passagem de parâmetro por valor ou por referência (*val* ou *ref*), <identificador do parâmetro> é o nome dado ao parâmetro de definição, e <tipo> é o tipo do valor associado ao parâmetro

Nota 1: para cada parâmetro especificado no cabeçalho, deve ser estabelecido o tipo de passagem (colocar **val** ou **ref** respectivamente para passagem por valor ou por referência) e o tipo do parâmetro associado ao valor armazenado.

Nota 2: a especificação de mais parâmetros deve ser precedida pela vírgula (",")

Nota 3: os parâmetros declarados numa sub-rotina podem ser utilizados como se fossem variáveis locais.

Exemplo 1:

```
procedimento exemplo1(val a:inteiro, ref b:inteiro);
```

Exemplo 2:

```
funcao exemplo2(val a:inteiro, val b:inteiro): real;
```

Para cada parâmetro de definição, deve haver um parâmetro de chamada, ou seja, na chamada da sub-rotina devem ser especificados tantos argumentos quantos existirem na sua declaração. O tipo de cada parâmetro de chamada deve estar de acordo com o tipo do parâmetro de definição.

Exemplo: suponha a declaração da sub-rotina a seguir:

```
funcao maior(val a:inteiro, val b:inteiro):inteiro;
```

```
inicio
```

```
  se (a>b)
```

```
    entao retorna (a);
```

```
    senao retorna (b);
```

```
  fimse;
```

```
fimfuncao;
```

```
inicio
```

```
  \na chamada da função dois parâmetros de chamada devem ser passados
```

```
  \como argumentos para serem associados aos parâmetros "a" e "b".
```

```
  \Esta associação é feita pela posição dos parâmetros na lista.
```

```
  escreva ("O maior número entre ", x, " e ", y, " é", maior(x,y));
```

```
  \nesta chamada de função, as variáveis "x" e "y" são associadas
```

```
  \aos parâmetros "a" e "b" respectivamente.
```

```
finalgoritmo.
```




T