UNIVERSIDADE FEDERAL DE SANTA CATARINA

PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA DE PRODUÇÃO

ÁREA DE CONCENTRAÇÃO: MÍDIA E CONHECIMENTO

ABC-Pro: Um Ambiente para Apoio ao Ensino da Lógica de Programação

Mestrando: Andrino Fernandes

Orientador: Fernando Álvaro Ostuni Gauthier

FLORIANÓPOLIS (SC) / JUNHO DE 2000.

UNIVERSIDADE FEDERAL DE SANTA CATARINA
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA DE PRODUÇÃO

ABC-Pro : Um Ambiente para Apoio ao Ensino da Lógica de Programação

ANDRINO FERNANDES

Dissertação apresentada à Universidade Federal de Santa Catarina para obtenção do grau de Mestre em Engenharia de Produção.

FLORIANÓPOLIS (SC) / JUNHO DE 2000.

ABC-Pro: Um Ambiente para Apoio ao Ensino da Lógica de Programação

ANDRINO FERNANDES

Esta dissertação foi julgada adequada para obtenção do título de "Mestre em Engenharia de Produção" aprovada em sua forma final pelo Programa de Pós-Graduação.

Profº Ricardo Miranda Barcia, Ph.D.
Coordenador do Curso de Pós-Graduação em Engenharia de Produção

Banca Examinadora:

Profº Fernando Álvaro Ostuni Gauthier, Dr.Orientador

Vânia Ribas Ulbricht, Dra.

Alejandro Martins Rodriguez, Dr.

FLORIANÓPOLIS (SC) / JUNHO DE 2000.

DEDICATÓRIA

Este trabalho é dedicado às pessoas que acreditam que as boas e grandes mudanças tem origem na educação.

AGRADECIMENTOS

- À Deus por mais este momento de alegria;
- Ao Professor Fernando Álvaro Ostuni Gauthier pela orientação e dedicação, principalmente nos momentos de angústia e incertezas;
- À Regina e Felipe pelo carinho, amor e paciência;
- Aos meus pais Andrino Genuíno Fernandes (*in memorian*) e Laura Julia Fernandes que iluminaram meu caminho com humildade e sabedoria, juntamente com os demais familiares;
- Aos amigos do NIS pelo apoio, ajuda e por acreditar na importância deste trabalho;
- Aos alunos do Curso Técnico de Informática que colaboraram com a pesquisa;
- Aos professores da banca examinadora, Dra. Vânia Ribas Ulbricht e Dr. Alejandro Martins Rodrigues pela colaboração;
- Ao CEFET/SC e à UFSC por todo suporte lógico e físico disponibilizado.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIAÇÕES

CEFET/SC	- Centro Federal de Educação Tecnológica de Santa Catarina
UFSC	- Universidade Federal de Santa Catarina
ETF/SC	- Escola Técnica Federal de Santa Catarina
FETESC	- Fundação do Ensino da Escola Técnica Federal de Santa Catarina
UNOESC	- Universidade do Oeste de Santa Catarina
NIS	- Núcleo de Informática e Sistemas
IBM	- International Business Machines
ENIAC	- Eletrical Numerical Integrator and Calculator
MIT	- Massachusetts Institut Tecnology
AF	- Autômato Finito
GLC	- Gramática Livre de Contexto
BNF	- Forma de Backus-Naur
GAS	- Gerador de Analisador Sintático
SE	- Sistema Especialista
RBC	- Raciocínio Baseado em Casos
POO	- Programação Orientada à Objetos
WWW	- Word Wide Web

LISTA DE GRÁFICOS

LISTA DE FOTOGRAFIAS

RESUMO

Este trabalho tem como principal objetivo o desenvolvimento e a utilização do ABC-Pro, um software para o apoio ao ensino da lógica de programação. O ponto fundamental do trabalho é proporcionar facilidades para implementação de programas através de uma representação algorítmica. O aspecto pedagógico também faz parte desta abordagem. O conteúdo descritivo faz uma explanação do domínio do problema e as teorias de aprendizagem. O ambiente permite tratamento de erros numa linguagem apropriada para os iniciantes da programação, bem como possibilita aos alunos os conceitos e definições sobre o conteúdo básico necessário para elaboração de programas. O compilador como núcleo principal do ambiente é a origem de todo o trabalho e devidamente relatado. A aplicação deste trabalho teve como público alvo os alunos do Curso Técnico de Informática do CEFET/SC. Espera-se com este trabalho fornecer uma contribuição ao aluno proporcionando o aprendizado da lógica de programação e para o professor um instrumento que auxilie o acompanhamento e gerenciamento do processo de desenvolvimento cognitivo do aluno.

ABSTRACT

This study has the main purpose of applying and developing the ABC-Pro. It is a software used to support logic programming teaching. The main focus of this study was to make easier the implementation of the program by means of an algorithm representation. Pedagogical issues were also involved in this approach. An explanation about the problem domain was introduced and learning theories were provided along the descriptive content. The environment allowed to deal with mistakes following an appropriate language that was considered suitable to programming beginners. It also allowed the introduction of concepts and definitions about basic content which were required in the development of logic programming. The compiler was seen as the main nucleous of the environment and it was the source of the whole study. The subjects were students of the Technical Course of Computing at CEFET/SC. There is a hope that this work may be seen as a contribution to the students who learned about logic programming and to the teachers as an instrument that may help tuttoring and management along the development of the student cognitive process.

CAPÍTULO I

1 - INTRODUÇÃO

1.1 - Apresentação

Com as novas tendências tecnológicas é imprescindível que estudos e pesquisas na área educacional busquem através da informática, alternativas que facilitem, e ao mesmo tempo, auxiliem o homem no processo ensino-aprendizagem.

Vive-se um processo crescente no uso de tecnologias aplicadas à educação, é importante ressaltar não só a contribuição dos recursos computacionais existentes, mas, também, a necessidade de estudar a forma ideal para sua aplicação. O computador surge como uma ferramenta de trabalho e de transmissão de conhecimentos que não pode deixar de ser considerado.

Neste contexto, as escolas e os educadores tem apenas reagido ao desenvolvimento tecnológico e às transformações sociais. O planejamento pedagógico tem sido orientado pela tecnologia para a utilização do computador em sala de aula, quando seria importante refletir cuidadosamente sobre as necessidades, objetivos e recursos disponibilizados para o suporte ao processo ensino-aprendizagem.

Enquanto a informática evolui rapidamente, as ferramentas pedagógicas se mostram em certos passos, insuficientes e, quando muito, pouco difundidas. As abordagens pedagógicas sempre se mostraram abertas quanto a utilização de meios que favorecessem o processo cognitivo. Mas, a utilização dessas ferramentas são, em muitos casos, aplicadas de forma ineficiente e estrategicamente mal elaboradas.

O computador presente na escola não significa necessariamente melhoria na qualidade do ensino. Cabe aos educadores contribuir no sentido de torná-los atraentes e dinâmicos a ponto de serem recursos promotor de mudanças construtivas das habilidades. A verdadeira função dos recursos educacionais não deve ser a de ensinar mas sim a de criar condições de aprendizagem (Hawkins, 1995).

A construção de ambientes educacionais computacionais devem proporcionar aos usuários uma forma lúdica de construir conceitos normalmente considerados de difícil compreensão, permitindo a apropriação e recriação do conhecimento de forma efetiva e natural, segundo Cardozo e Ramos (1995).

A adoção de tecnologias permite potencializar formas de interação. Entretanto, novas estratégias pedagógicas e formas de construção e apropriação devem ser desenvolvidas para que a transformação no processo de aprendizagem seja qualitativa e não meramente quantitativa.

A tarefa dos educadores, cita Moran (1993), é educar para novas experiências, educar para a mudança, para o não previsível, educar para a autonomia, para a autenticidade, para desenvolver o mais plenamente possível todas as potencialidades individuais.

Cabe à escola e aos educadores a apropriação dessas novas tecnologias educacionais tornando o ato de aprender mais interativo, concreto e cooperativo.

1.2 - Definição do Tema

Professores e outros estão procurando meios para melhorar a aprendizagem. Pretende-se encontrar meios mais eficazes. As soluções, em grande parte, dependem do conhecimento das causas da aprendizagem humana. Quanto mais conhecer os fatores que ajudam alguém a aprender, maior os benefícios para trazer melhorias à sua aprendizagem (Howe, 1986).

Diante disso, busca-se estudar o contexto no qual o aluno está inserido, suas características cognitivas e o processo de aprendizagem da lógica de programação.

Ao trabalhar no Curso Técnico de Informática do Centro de Formação Tecnológica de Santa Catarina (CEFET/SC), mais especificamente com a disciplina de Linguagem de Programação, surgiu a necessidade e o interesse em desenvolver uma ferramenta que propiciasse o aprendizado da disciplina de forma que o aluno pudesse colocar em prática seus conhecimentos de lógica na elaboração de programas sem precisar abstrair seu conhecimento através de algoritmos, nem tão pouco se defrontar com uma linguagem de programação que, em algumas vezes, acaba tornando o processo de aprendizagem mais trabalhoso.

Acreditando que o processo de aprendizagem possa ser otimizado, surgiu a necessidade de desenvolver um ambiente onde o aluno pudesse aplicar seus conhecimentos de forma a utilizar uma linguagem que faça parte de seu cotidiano, isto é, um ambiente em que o aluno pudesse escrever as instruções lógicas para resolução de um problema de forma algorítmica e ao mesmo tempo promover uma análise do programa desenvolvido. Além disso, para complementar a validação do processo cognitivo, um ambiente que proporcione um tratamento e orientação adequados à eventuais erros.

Esta ferramenta tende a gerar incentivos que afetarão o aprendizado diário, ou aumentando a probabilidade de que um indivíduo preste mais atenção ou

encorajando-o a se engajar num processo intensivo dos itens a serem aprendidos.

1.3 - O Contexto do Problema

Quando se está aprendendo novos conteúdos, ajusta-se naturalmente nossa estrutura cognitiva aliada a um processo de ensino-aprendizagem para que haja uma acomodação do conhecimento.

> "A eficácia da aprendizagem não está somente em aumentar o tamanho do conhecimento do indivíduo. Também é importante o fato de que há mudanças organizacionais refletindo na maneira em que a informação está estruturada. As mudanças na estrutura do que a pessoa sabe, apresenta implicações para uma aprendizagem futura." [Howe, 1986, pg. 80]

No processo de aprendizagem de desenvolvimento de programas de computador o aluno aprende inicialmente como desenvolver algoritmos para assimilação dos conceitos que originarão fundamentos para aplicação em linguagens de programação. No entanto, as dificuldades se manifestam de forma a retardar e, até mesmo, desmotivar o aluno no processo de aprendizagem.

Considera-se as seguintes dificuldades:

- Novo conhecimento: elaborar algoritmos envolve, em sua maioria, conteúdos, até então nunca vistos pelos alunos, ou seja, quase não existe busca de experiências passadas para reforçar do novo conhecimento;
- Abstração do conhecimento: no processo de aprendizagem tradicional, os alunos aprendem inicialmente a compor algoritmos, isto é, descrever os passos para a realização de uma tarefa. Isto implica em resoluções sem saber de forma autônoma se o algoritmo está correto ou não.
- Linguagens de programação: é o resultado prático dos algoritmos. No entanto, na aplicação deste recurso, é preciso converter as instruções escritas em algoritmo para uma linguagem em questão. Consiste num dos objetivos da disciplina.

- Gramática inconsistente: para concretizar o abstrato na utilização de linguagens de programação é preciso aprender as formas e regras na qual as linguagens estão estabelecidas, normalmente, diferentes dos algoritmos.
- Mensagens de erro: as formas e regras da linguagem são analisadas no processo de compilação do programa. A decorrência deste fato, aliado aos problemas anteriores, acabam, naturalmente, gerando erros. Estes erros são apresentados de forma extremamente objetiva, não possibilitando de imediato sua elucidação.

As conseqüências das dificuldades apresentadas acarretam o não-entendimento, provocando antipatia pela disciplina comprometendo o avanço do aprendizado. A desmotivação neste processo, acaba sendo uma conseqüência natural.

1.4 - Objetivo do Trabalho

O objetivo geral do trabalho é propor e implementar um ambiente computacional de apoio no processo de ensino-aprendizagem de lógica de programação.

Como objetivos específicos tem-se:

1. abordar as metodologias de ensino de lógica de programação;
2. estudar as teorias pedagógicas possíveis de serem utilizados no processo da lógica de programação;
3. apresentar a teoria computacional necessária para análise de programas;
4. propor uma arquitetura para o ambiente;
5. implementar o sistema proposto;
6. aplicar o sistema proposto com a finalidade de uma avaliação preliminar.

1.5 - Justificativa

É incontestável a utilização da informática em todas as atividades profissionais e sociais, e o fascínio que esta exerce nos jovens. Portanto, se uma das finalidades da escola é preparar a integração dos alunos na vida ativa, torna-se

cada vez mais importante a utilização do computador como uma realidade no processo de ensino-aprendizagem.

Baseado nisto, resta saber qual a melhor forma de explorar esta tecnologia como uma ferramenta de auxílio na construção das estruturas cognitivas do aluno durante o aprendizado.

Em geral, muitos *softwares* educacionais refletem uma concepção inadequada do processo de ensino-aprendizagem: a de que ensinar é a mera transmissão de conteúdos, fazendo com que o aluno continue sendo um agente passivo como ainda ocorre no ensino tradicional.

Uma outra questão fundamental à criação de software educacional é a necessidade de uma formalização teórica. Seguindo a abordagem construtivista de Jean Piaget, o aluno deve ser vislumbrado não como um mero repositório de conhecimento, mas como um agente construtor deste conhecimento. Ele deve ser capaz de descobrir os conhecimentos intrínsecos aos objetos que manipula e não receber simplesmente estes conhecimentos prontos.

Pretendemos desenvolver recursos possibilitando ao modelo proposto uma futura utilização no ensino à distância.

1.6 - Estrutura do Trabalho

O trabalho foi desenvolvido em base bibliográfica, selecionado da literatura as contribuições sobre construção de compiladores para concepção e desenvolvimento do produto e da pedagogia e psicologia no processo cognitivo em ambientes de aprendizagem.

A revisão bibliográfica foi feita na literatura especializada, artigos impressos e materiais disponibilizados na internet.

O trabalho apresenta a seguinte estrutura:

No capítulo 2, são apresentadas as teorias da aprendizagem, suas aplicações nos ambientes informatizados e o processo pedagógico que envolve o ensino da lógica de programação.

No capítulo 3, é apresentado o processo de desenvolvimento de compiladores, sua estrutura, etapas de desenvolvimento e aspectos fundamentais quanto ao planejamento de compiladores.

No capítulo 4, é apresentada a proposta de estrutura para um ambiente de aprendizagem.

No capítulo 5, é apresentada a implementação do ambiente de aprendizagem.

No capítulo 6, é apresentada uma visão geral da instituição, sua concepção pedagógica, a importância da disciplina Programação no Curso Técnico de Informática e a aplicação desenvolvida com os alunos do Curso Técnico de Informática.

No capítulo 7, são apresentadas as conclusões e recomendações para trabalhos futuros.

CAPÍTULO II

2 – ABORDAGEM PEDAGÓGICA NO ENSINO DE PROGRAMAÇÃO

2.1- A Educação

A educação tem por objetivo uma aprendizagem que abranja conceitos e experiências, que libere o aluno para a auto-aprendizagem criando condições para que os alunos possam tornar-se pessoas de iniciativa, de responsabilidade, de autodeterminação, de discernimento, que saibam se aplicar, para aprender fatos novos para a solução de seus problemas, utilizando sua própria experiência.

O processo educativo pode ser concebido de várias formas, é um fenômeno humano, histórico e multidimensional. Nele estão presentes os aspectos humanos, técnicos, cognitivos, emocionais, os sócio-políticos e os culturais. Estes aspectos requerem múltiplas implicações e relações. De acordo com determinada teoria/proposta ou abordagem do processo ensino-aprendizagem, privilegia-se um ou outro aspecto do processo educacional (Nunes, 1999).

> "A educação é uma prática humana direcionada por uma determinada concepção teórica. A prática pedagógica está articulada com uma pedagogia, que nada mais é que uma concepção filosófica da educação. Tal concepção ordena os elementos que direcionam a prática educacional." [Luckesi, 1991, p.21]

Saviani [apud Nunes, 1999, p.46] acrescenta:

> "... o objeto da educação diz respeito, de um lado, à identificação dos elementos culturais que precisam ser assimilados pelos indivíduos da espécie humana para que eles se tornem humanos e, de outro lado e concomitantemente, à descoberta das formas mais adequadas para atingir esse objetivo."

Paulo Freire [1971, p.49] afirma que:

> "A questão da educação apresenta-se como uma relação entre o professor e aluno mediada pelo conteúdo. Dessa forma, o conteúdo não é absolutamente secundário, nem o ponto central: é o elemento que permite a relação educativa entre educador e educando."

E acrescenta:

> "Neste novo movimento estamos cada vez mais deixando de lado o modelo de aulas expositivas e abstratas por aulas práticas e vivenciadas, com forte tendência a uma educação mais criativa e libertadora e não meramente reprodutora."

A educação necessita de pressupostos, de conceitos que fundamentem e orientem seus caminhos tendo em vista que ela não se manifesta como um fim, mas sim como um instrumento de manutenção ou transformação social. A pedagogia pressupõem um direcionamento filosófico, este garante a compreensão dos valores que, hoje, direcionam a prática educacional e os valores que deverão orientá-la.

2.2- Aprendizagem

Desde que a psicologia se arvorou em ciência que estuda os fenômenos simples e complexos de integração e organização do comportamento, a psicologia da aprendizagem assumiu uma importância muito grande, pois o comportamento é decorrência das modificações que a aprendizagem oportuniza.

A psicologia da aprendizagem é um campo específico de trabalho, dentro da psicologia, que abarca os fenômenos relacionados com a modificação de comportamento.

Antigamente a aprendizagem era compreendida como aquisição de informação, em um sentido predominantemente cognitivo. Na Idade Média entendia-se "aprender" como: fixar na memória ou conhecer.

Muitas outras idéias sobre aprendizagem têm sido expressas em definições que, na maioria das vezes mencionam apenas o comportamento observável do qual se infere, se houve ou não alguma aprendizagem, mas dificilmente se preocupam com qualquer processo interno de mudança. Em Duarte (1981, pg.448-450) temos alguns exemplos:

> "A aprendizagem, como mudança de uma realização numa certa direção, consiste em criar sistemas de traços de um tipo particular, consolidá-los e torná-los mais e mais disponíveis, tanto em situações repetidas como em novas situações." [KOFFKA, 1935]

> "Aprendizagem é a aquisição de uma nova resposta ou a execução aumentada de antigas respostas." [UNDERWOOD,1949]

> "Ocorre aprendizagem quando as informações provenientes do mundo externo e transmitidas pelo sistema nervoso causam uma mudança mais ou menos permanente do comportamento futuro." [KIMBLE,1961]

> "Aprendizagem é um processo ou operação, inferida de mudanças relativamente permanentes no comportamento, resultantes de uma prática." [KLAUSMEIER,1977]

Hoje, a aprendizagem é compreendida como determinante de toda a modificação de comportamento que resulta da experiência, ou como determinante de um potencial de respostas através do qual há predisposição para modificação de comportamento.

Segundo Leif (1976), do ponto de vista pedagógico, aprendizagem pode ser: "aquisição de conhecimentos pela experiência ou atividades intelectual, geralmente com o fim de se poder realizá-los ou pô-los em prática; aquisição da capacidade de fazer, praticar ou empreender um ato, ação ou qualquer coisa; aquisição da capacidade técnica de exercer uma profissão; ensino dado a alguém, especialmente a um aluno, com a finalidade de fazê-lo atingir um objetivo". Entretanto, o conceito de aprendizagem é um conceito psicológico e, como tal, deve ser analisado à luz das teorias psicológicas, as quais, nos últimos anos, têm ampliado os conhecimentos nesta área, de modo a sistematizar diversas teorias, agrupadas especialmente em torno do conexionismo e do cognitivismo.

O estudo da aprendizagem nos faz entender que o processo do desenvolvimento é o elemento básico para a adaptação, é tendência particular do ser humano, explicita a estrutura e dinâmica da personalidade humana e amplia e diversifica a motivação humana. A aprendizagem oferece subsídios para a elaboração de teorias de ensino.

2.2.1- Teorias de Aprendizagem

Há uma série de maneiras de classificar as teorias da aprendizagem que tem importância na atualidade. Como objetivo, uma diferença assume especial relevo: a diferença entre as teorias conexionistas e as teorias cognitivas.

As interpretações conexionistas da aprendizagem, por mais que apresentem diferenças entre si, encontram um ponto de concordância quando tratam a aprendizagem como uma questão de conexões entre estímulos e respostas. Entretanto, sempre se focalizam as respostas que ocorrem, os estímulos que as provocam e as maneiras como as experiências modificam essas relações entre estímulos e respostas.

As interpretações cognitivas ocupam-se das cognições (percepções ou atitudes ou crenças) que os indivíduos tem a respeito do ambiente, e das formas como essas cognições determinam o comportamento. Nessas interpretações, a

aprendizagem é o estudo das formas segundo as quais as cognições são modificadas pela experiência.

A preferência entre conexionismo ou cognitivismo depende do tipo de aprendizagem que mais interessar. Em estudos do condicionamento, uma interpretação conexionista pode ajustar-se melhor às necessidades, enquanto que em solução de problemas complexos, uma interpretação cognitiva pode ser a mais útil.

As diferenças filosóficas inclinam as pessoas na direção de uma ou de outra forma de interpretação mas, a distinção entre as teorias conexionistas e teorias cognitivas não é, naturalmente, uma questão de tudo-ou-nada; há numerosas posições intermediárias e combinações.

A teoria da aprendizagem conexionista enfatizou os comportamentos mensuráveis e observáveis bem como suas modificações. Esta teoria forneceu os fundamentos dos primeiros projetos de tecnologia instrucional baseada em computador.

É tratado com maior ênfase as abordagens da teoria cognitiva pois, é a referência pedagógica para a ferramenta desenvolvida proporcionando flexibilidade dos processos intelectuais, permitindo ao aluno resolução de problemas nos mais variados níveis de complexidade e através disso, construir seu conhecimento.

2.2.2 - Abordagem Construtivista

O construtivismo é uma postura filosófica que parte do princípio de que o desenvolvimento da inteligência é determinado pelas ações mútuas entre o indivíduo e o meio. Essa concepção teórica determina que o homem não nasce inteligente, mas também não é passivo sob a influência do meio. O indivíduo responde aos estímulos externos agindo sobre eles para construir e organizar o seu próprio conhecimento, de maneira cada vez mais elaborada (Bolzan, 1998).

Segundo Bolzan (1998), a estrutura é construída na medida em que o indivíduo vai agindo sobre o meio físico ou social, transformando-o em algo que ele não era. Aprender uma outra língua, por exemplo, é construir estruturas, podendo assimilar qualquer conteúdo no nível de abstração. Becker (1997) considera que o conhecimento é uma construção individual. "O sujeito humano é um projeto a ser construído por ele mesmo". O conhecimento nessa postura filosófica é construído pela ação do sujeito, na relação hereditariedade e meio.

Para Vygotsky (1991), o conhecimento é visto como um resultado da construção do próprio indivíduo, através da interação do sujeito com o mundo, considerando os fatores biológicos, experiências físicas, a troca social e os processos de equilíbrio e desequilíbrio nessa construção. De acordo com essas idéias, o indivíduo é o motor ativo e coordenador do seu próprio desenvolvimento.

A respeito das contribuições de Vygotsky, Oliveira (1995) destaca, inicialmente, que o homem não nasce pronto, ele se constrói ao longo de um percurso de desenvolvimento psicológico no qual a interação com o grupo e o outro social é fundamental. Ele vai reconstruir e desenvolver, em nível individual, o material recebido do contexto sócio-cultural. A transmissão do contexto, portanto, não se faz de maneira mecânica, o indivíduo negocia constantemente as informações recebidas.

A teoria construtivista propõe que o aluno participe ativamente do próprio aprendizado, mediante a experimentação, a pesquisa em grupo, o estímulo à dúvida e o desenvolvimento do raciocínio, entre outros procedimentos. Esta teoria, rejeita a apresentação de conhecimentos prontos ao estudante, como um "prato feito". Daí o termo "construtivismo", pelo qual se procura indicar que uma pessoa aprende melhor quando toma parte de forma direta na construção do conhecimento que adquire.

Piaget estudou as "engrenagens" da inteligência, do nascimento à maturidade do ser humano, onde decifrou sucessivos degraus na evolução do raciocínio, ou seja, em como a inteligência humana se desenvolve, atribuindo o nome de

construtivismo. Outras correntes também empenhadas em explicar sobre o desenvolvimento da inteligência surgiram: o empirismo e o racionalismo.

Essas três correntes divergem quanto à relação entre meio ambiente e inteligência. O empirismo é uma concepção teórica que explica que o desenvolvimento da inteligência é determinado pelo meio ambiente e não pelo sujeito, ou seja, o desenvolvimento intelectual é submetido às forças do meio, vem de fora para dentro, a inteligência vai se modelando através dos estímulos externos e não do indivíduo. Já o racionalismo é uma concepção teórica que concebe o desenvolvimento intelectual determinado pelo indivíduo e não pelo meio. A inteligência já nasce pré-moldada com o indivíduo, sendo reorganizada pelas percepções da realidade na medida em que o ser humano vai amadurecendo. Os estímulos externos não são considerados e sim as capacidades que são inerentes ao indivíduo.

Dentre as teorias contemporâneas de aprendizagem, o trabalho de Piaget, devido à pertinência com que suas preocupações epistemológicas, biológicas e lógico-matemáticas, têm sido difundido e aplicado para o ambiente educacional, em especial na didática e em alguns dos ambientes de aprendizagem auxiliados por computador.

Piaget estudou detalhadamente e explicou na sua teoria chamada de Epistemologia Genética ou Teoria Psicogenética como a inteligência vai se construindo desde o nascimento. Esta é a concepção construtivista mais conhecida.

Enquanto que no construtivismo é a pessoa que constrói o seu próprio conhecimento, nas teorias empiristas e racionalistas reduzem o desenvolvimento intelectual somente à força do meio ou à ação do indivíduo. Piaget aborda que a questão do desenvolvimento da inteligência está em manter um equilíbrio dinâmico com o meio ambiente. Quando o equilíbrio se rompe, o indivíduo age sobre o que o afetou (um som, uma imagem, uma informação) buscando se reequilibrar. Esse equilíbrio é feito através da adaptação e organização (Bolzan, 1998).

Segundo Piaget (1987), o desenvolvimento do indivíduo passa por sucessivas equilibrações que foram caracterizadas em quatro estágios distintos mas contínuos que descrevem o desenvolvimento da inteligência. São eles:

- sensório-motor (0 a 2 anos);
- pré-operatório (2 a 7 anos);
- operatório-concreto (7 a 11 anos) e
- operatório-formal (11 anos em diante).

No sensório-motor, a inteligência é prática. A partir de reflexos neurológicos, começa a construir esquemas de ação para assimilar mentalmente o meio. No estágio pré-operatório, torna-se capaz de representar mentalmente pessoas e situações. Tem percepção global, não atenta para detalhes. É centrada em si mesma, não tem noção de abstrato. Já na fase operatório-concreto, é capaz de relacionar diferentes aspectos e abstrair dados da realidade. A criança nessa fase depende ainda do mundo concreto para chegar à abstração. O estágio operatório-formal permite que a representação tenha abstração total, sendo capaz de pensar em todas as relações possíveis logicamente.

O ponto de equilíbrio do estágio operatório-formal situa-se na fase da adolescência. Nesta fase o pensamento se torna livre das limitações da realidade concreta. Surge uma nova forma de raciocinar, que não incide exclusivamente sobre o concreto mas sobre as hipóteses. A partir deste estágio a criança pode pensar de modo mais lógico e correto mesmo com um conteúdo de pensamento incompatível com o real. O adolescente livre do concretismo pode trabalhar com a realidade possível, com hipóteses dedutivas, e, neste ponto começa a abstração e todos os desencadeamentos que ela pode oferecer.

Sob a perspectiva piagetiana, o pensamento é a base em que se fundamenta a aprendizagem, e esta é uma construção centrada na pessoa que a realiza. Piaget, revelou que o ser humano tem uma pré-disposição para pensar, julgar, argumentar com bases racionais, e necessita desenvolver estas potencialidades no decorrer da vida. Portanto, é importante ressaltar que o

conhecimento é produzido na interação com objetos do ambiente, propiciando o desenvolvimento de esquemas mentais e, por conseguinte, o aprendizado.

Segundo Howe (1986), Bartleett, que foi um dos primeiros psicólogos a desenvolver o ponto de vista de que o conhecimento estava representado em formas de estruturas mentais inconscientes, considerou que esquemas diferentes não se representam separadamente, mas como uma massa unitária. O que é mais importante, ele considerou que, depois de uma exposição a muitos acontecimentos e itens de informação, a mente retém um tipo de "representação cognitiva genérica", ou esquema, contendo as estruturas essenciais ou os elementos que uma entrada de um grande número de informações tem em comum. Alguns psicólogos modernos preferem usar a palavra *script*, mas, com o mesmo significado. Portanto, esquemas ou *scripts* é definido como: uma organização ativa de reações ou experiências passadas, supostas a operar de uma maneira bem adaptada.

Nos últimos anos, uma ênfase considerável tem sido dada à importância dos esquemas ou *scripts* na aprendizagem e compreensão. Eles fornecem bases de referência que nos ajudam a compreender e lembrar acontecimentos experienciados, servindo como estrutura do conhecimento organizado, ao qual novos eventos podem ser relacionados.

Segundo Piaget (1967), um esquema não é aprendido nem memorizado, mas construído e conservado, sendo imprescindível à aprendizagem. Esta, por sua vez, poderá ser entendida, a partir daí, como aquisição ou conservação de conteúdos específicos, acrescidos de mecanismos coordenadores que dão coerência àqueles conteúdos.

Para Howe (1986), a implicação para a aprendizagem escolar, é que a possibilidade de o aluno reter informação sobre acontecimentos e atividades pode ser aumentada, colocando a informação no contexto, no qual um esquema existente já está disponível.

Para Piaget [apud Casas, 1998, p.86]:

> "As estruturas específicas para o ato de conhecer são construídas como resultado de um processo de equilibração em que, num processo de adaptação progressiva, através de suas ações, o organismo mantém uma troca com o meio. Este processo de adaptação não significa, segundo o autor, um ajustamento passivo ao meio, em função da sobrevivência do sujeito, mas, sim, um processo pelo qual o indivíduo se transforma em função do meio e transforma o próprio meio à medida que o assimila por suas ações e operações."

Esse processo de adaptação e readaptação é explicado por Piaget como mecanismo duplo, de assimilação e acomodação. Pela assimilação, os objetos são incorporados aos esquemas de ação do sujeito. Desse processo resultam alterações na própria organização mental do indivíduo, que acaba modificando-se em decorrência de um esforço para proceder as novas assimilações. A essa modificação, Piaget denomina de acomodação dos esquemas.

A importância da noção de assimilação na teoria piagetiana é dupla: de um lado implica a significação (uma vez que todo conhecimento se prende a significações, desde os mais simples índices e sinais percebidos até a função simbólica da linguagem humana, por exemplo) e de outro exprime um fato fundamental: todo conhecimento está ligado a uma ação e, conhecer um objeto ou um fato é ser capaz de utilizá-lo assimilando-o a esquemas particulares (Piaget, 1967). Conhecer não consiste, de fato, em copiar o real, mas em agir sobre ele e transformá-lo (em aparência ou em realidade), de maneira a compreendê-lo em função dos sistemas de transformação aos quais estão ligadas estas ações.

A aprendizagem é um processo no qual experiências fomentam modificação do comportamento e aquisição de hábitos. Segundo Wechsler (1993), Piaget ao estudar a gênese do desenvolvimento da inteligência, demonstrou a importância da maturação do sistema nervoso, da ação sobre os objetos e dos fatores sociais como variáveis influenciantes na compreensão do processo

intelectual. Em seus estudos, demonstrou como os processos de assimilação e acomodação de novos conhecimentos se incorpora à estrutura do pensamento.

A assimilação e a acomodação são os motores da aprendizagem. A adaptação intelectual ocorre quando há o equilíbrio de ambas. Ulbricht (1997) coloca que a aquisição do conhecimento cognitivo ocorre sempre que um novo dado é assimilado à estrutura mental existente que, ao fazer esta acomodação modifica-se, permitindo um processo contínuo de renovação interna.

Pela assimilação, justificam-se as mudanças quantitativas do indivíduo, seu crescimento intelectual mediante a incorporação de elementos do meio a si próprio. Pela acomodação, as mudanças qualitativas de desenvolvimento modificam os esquemas existentes em função das características da nova situação; juntas justificam a adaptação intelectual e o desenvolvimento das estruturas cognitivas (Ulbricht, 1997).

O desenvolvimento das estruturas mentais segue uma construção semelhante aos estudos da lógica, ou seja, o desenvolvimento da inteligência em seus sucessivos estágios segue uma seqüência coerente, podendo ser descrita em suas diversas etapas.

O construtivismo enfatiza a importância do erro não como um tropeço, mas como um trampolim na rota da aprendizagem. O erro passa a ter um caráter "construtivo", isto é, serve como propulsor para se buscar a conclusão correta.

A teoria construtivista, particularmente a de Piaget, é o pressuposto teórico do movimento que gerou o *software* LOGO.

Construtivismo não finaliza, nem esgota, aquilo que se deve saber sobre educação, não é começo e nem fim, pois o desenvolvimento de novas pesquisas, ampliam a cada dia as fronteiras e limites do conhecimento pedagógico.

2.2.2.1 – O Professor

O educador é um pesquisador em ensino, procurando alimentar esta teoria a partir das questões advindas da prática escolar.

> "Dos professores, espera-se que eles sejam especialistas na aprendizagem e que possam usar os seus conhecimentos e habilidades para ajudar os outros a aprender. Qualquer avanço realizado no sentido de se aumentar a compreensão dos processos da aprendizagem humana, beneficia o professor, desde que esse progresso corresponda a conhecimentos práticos, de como as crianças adquirem os tipos de habilidades necessárias à aprendizagem na escola." [Howe, 1986, p.11]

No construtivismo, o professor é aquele que questiona constantemente o aluno colocando desafios. Com perguntas ele instiga o "espírito descobridor" com o objetivo de otimizar a aprendizagem. Se o professor conhece o aluno através dos questionamentos, ele terá sempre perguntas a fazer e o aluno estará assim construindo suas estruturas, desenvolvendo a sua inteligência. Essa comunicação é dita por Paulo Freire de teoria dialógica e é fundamental no desenvolvimento intelectual. Toda vez que se pensa sobre algo, se constrói conhecimento e isto significa refazer aquela estrutura que estava ali até aquele momento.

A função do professor não é portanto depositar informações em grande quantidade e das mais diversas formas possíveis. Segundo Becker (1997), "Professor é alguém que desafia o aluno a reconstruir suas estruturas, e quem sabe, a construir novas estruturas. E para isso ele tem que colocar o aluno diante de si mesmo, perante a sua história de ações". O professor deve trabalhar o conteúdo, não de maneira quantitativa, mas qualitativamente, procurar resolver problemas e a tomada de decisões, colocando a estrutura cognitiva em ação. Ensinar não é apenas fazer com que os alunos adquiram pré-requisitos na memória, acumulando passivamente as informações.

Nesse sentido uma ação docente construtivista pautar-se-á nas condições concretas do aluno, no conhecimento dos períodos de seu desenvolvimento em

relação aos esquemas de elaboração mental, no respeito a sua individualidade dentro do contexto grupal em que está inserido.

O papel do professor, aos poucos, está cedendo lugar ao de um guia no universo do conhecimento. Dessa forma, o professor deverá ser o orientador, o coordenador e o incentivador do aprimoramento das funções de pensamento. Sua tarefa será a de estimular os alunos a navegar pelo conhecimento e a realizar suas próprias descobertas. Com o computador, o relacionamento entre professores e alunos tende a ser mais descontraído e interpessoal.

2.2.2.2 – A Motivação

A grande quantidade de bibliografia sobre o tema demonstra que o interesse pela motivação tem levado pesquisadores a buscarem respostas para essa variável imprescindível, pois a aprendizagem proporcionando a modificação do comportamento satisfaz a motivos individuais para que ocorra o desenvolvimento da aprendizagem.

Para Campos (apud Bolzan, 1998, p.45):

> "A motivação é um processo interior, individual, que deflagra, mantém e dirige o comportamento. Implica num estado de tensão energética, resultante da atuação de fortes motivos que impelem o sujeito a agir com certo grau de intensidade e empenho".

Motivar é levar o aluno a empenhar-se em aprender, consistindo em proporcionar uma situação que induza o aluno à um esforço intelectual. A motivação é fator decisivo no processo ensino-aprendizagem e, toda a ação didática não será suficiente para que a aprendizagem ocorra se o aluno não estiver motivado, se não estiver disposto a despender esforços por aprender.

De acordo com Vilarinho (apud Bolzan, 1998, p.46):

> "Seja numa aprendizagem motora, ou numa que envolve a compreensão de relações e conceitos ou a apreensão de

> valores, só haverá aprendizado quando houver atividade do aprendiz, que por sua vez necessita de motivos para despertá-lo à ação".

O estímulo que gera a motivação é determinado pela atuação do professor deflagrado através do incentivo.

Para muitos professores, motivar é despertar o interesse. No entanto, a definição de interesse para Campos (1972) é a "atração emotiva exercida por um objeto ideal sobre a individualidade consciente. O interesse pode ser imediato (subjetivo) quando se liga a um objeto atual, implicando na relação com a própria atividade, ou pode ser mediato (objetivo), quando se liga a um objeto ideal, implicando na relação com o objetivo para a qual a atividade se dirige" Neste sentido, o professor pode incentivar o aluno a despertar os motivos para a aprendizagem.

A incentivo para Vilarinho (1986), implica na proposição de situações de modo a deflagrar no psiquismo do sujeito as fontes de energia interior (motivos), que o levarão à ação com empenho e entusiasmo. Incentivar é manipular as condições externas ao sujeito, de forma a despertar no aprendiz a motivação que mantém o processo de aprendizagem.

A incentivo pode ser intrínseca ou extrínseca. A intrínseca estimula o aluno a estudar uma disciplina pelo próprio valor que a disciplina apresenta. A extrínseca estimula o aluno a estudar uma disciplina pelas vantagens que ela pode proporcionar na vida do aluno.

No processo de motivação do aluno, é preciso que haja conecção entre os objetivos do conteúdo e o interesse do aluno, que este sinta a necessidade e a importância de aprender, isso o levará a esforçar-se.

A compreensão e o emprego correto das técnicas de motivação despertam o interesse da classe, elevam a sua moral, acarretam a aprendizagem eficaz dos alunos e dão-lhes uma sensação de realização.

2.2.2.3 – A Pesquisa

A importância da aprendizagem através da pesquisa é fundamental na prática de um ensino construtivista. Através dela é possível manejar e produzir conhecimento, possibilitando abstração reflexionante, isto pode produzir independência em pensamentos e ações. Em decorrência disso, a educação escolar pode contribuir grandemente na formação de pessoas com senso crítico da realidade.

O ensino deve buscar a aplicação da essência da pesquisa, isto é, o questionamento sistemático procurando intervir na realidade, ou o diálogo crítico permanente com a realidade, em sentido teórico e prático. A educação seguindo este princípio prepara o cidadão não para reproduzir o conhecimento aprendido em sala de aula, mas para a faculdade de julgar e agir diante das adaptações impostas pelas mutações da vida social e profissional.

2.2.3 - As Inteligências Múltiplas

Na perspectiva de muitos educadores, o conhecimento não é fragmentado, mas sim interligado. Gardner (1995) afirma que conhecemos através de um "sistema de inteligências interconectadas e, em parte, independentes, localizadas em regiões diferentes do nosso cérebro, com pesos diferentes para cada indivíduo e para cada cultura".

Segundo Gardner (1995), essa ampla variedade de inteligências humanas conduz a nova visão de educação, a qual o autor chama de "educação centrada no indivíduo". Essa nova perspectiva de educação equivale a uma visão pluralista da mente, reconhecendo que as pessoas têm forças cognitivas diferenciadas. O ensino baseado no computador converge para esse pensamento, pois a descoberta faz parte desse aprendizado.

Gardner conduziu a sua pesquisa baseada nos estilos cognitivos diferenciados apresentados pelos indivíduos. A sua teoria é apresentada sob a forma das "inteligências múltiplas". Todos nós possuímos as inteligências ou habilidades,

porém com pesos diferentes. Segundo o autor, a inteligência ou habilidade lingüística é aquela em que se manifesta o gosto pela leitura, escrita, ouvir e contar histórias e que facilita a compreensão através das palavras faladas ou escritas. Uma outra inteligência ou habilidade é apresentada pela lógico-matemática, que pode estruturar, organizar e sintetizar os conteúdos da vida cotidiana e a encontrar ordem no caos. Uma outra é apresentada sob a forma de inteligência espacial, que está em trabalhar com imagens, capacidade de visualizar espacialmente as fotos, as imagens, o visual. A sensibilidade para ambientes musicais e melodias está na inteligência musical, onde o aprendizado é favorecido através do som. A inteligência cinestésico-corporal é aquela onde a informação chega mais rápido através do movimento e do toque. Nessa abordagem, a aprendizagem é mais rápida quando o indivíduo está se movimentando.

As outras inteligências ou habilidades são complementares: uma é a intrapessoal e a outra, a interpessoal. Na intrapessoal predomina a busca da auto-realização. Na interpessoal aprende-se melhor através da interação com os outros. Gardner vem pesquisando também a inteligência ou habilidades teológica e ambiental.

As particularidades dos estudantes são destacadas por Gardner (1994, p.293), quando ele alerta:

> "(...) é uma suposição essencial deste estudo que os indivíduos não são todos iguais em seus potenciais cognitivos e em estilos intelectuais e que a educação pode ser mais adequadamente efetuada se for talhada para as capacidades e necessidades dos indivíduos particulares envolvidos. De fato, o custo de tentar tratar todos os indivíduos da mesma forma ou de tentar transmitir conhecimentos para indivíduos de maneira inapropriada ao seus modos de aprendizado pode ser grande."

Gardner demonstra com a sua teoria, que todo ser humano é capaz de chegar ao conhecimento, porém com intensidade diferentes, pois a aprendizagem muda de pessoa para pessoa. Algumas pessoas têm mais facilidade de aprender através da fala, outros através de cálculos, ou através da música ou do movimento e também da cooperação entre as pessoas.

Ao ler os trabalhos de Gardner nota-se que seu núcleo central não está no número que podem ser associadas à inteligência mas sim, fundamentalmente, no caráter múltiplo que a inteligência apresenta e na possibilidade de podermos olhar para as manifestações da inteligência, não mais sob a perspectiva de uma grandeza a ser medida ou como um conjunto de habilidades isoladas, mas como uma teia de relações que se tece entre todas as dimensões que se estabelecem nas possibilidades de manifestação da inteligência.

Todos tem os mesmos instrumentos para chegar ao conhecimento, mas não os utilizam com a mesma intensidade. Normalmente, os processos educacionais se baseiam, quase exclusivamente, no desenvolvimento da inteligência lingüistica e da lógico-matemática, deixando de lado as outras formas de acesso ao conhecimento (Gardner, 1985).

Na opinião do pesquisador, sua teoria se contrapõe a esse modo de pensar porque pluraliza o conceito tradicional. Para Gardner e seus colaboradores, uma inteligência implica na capacidade de resolver problemas ou elaborar produtos que são importantes num determinado ambiente ou comunidade cultural. A capacidade de resolver problemas permite à pessoa abordar uma situação em que um objetivo deve ser atingido e localizar a rota adequada para esse produto.

2.3 - O Computador no Ensino

Vive-se um momento em que não dá mais para ignorar ou deixar de lado a influência dos computadores sobre as pessoas. Não importa a classe social ou a idade. Esta máquina exerce um fascínio sobre todos, atraindo a atenção e possibilitando situações de aprendizagem. A intervenção psicopedagógica na escola adquire nova amplitude quando inclui o computador como uma possibilidade de construção de conhecimento na escola.

Muito se tem escrito sobre a informática na escola, mas há pouco consenso entre os diversos autores e educadores sobre o valor do uso dessa tecnologia em relação aos ganhos que ele pode trazer aos alunos.

O uso do computador, enquanto instrumento tecnológico, na educação está sempre associado a milagres ou a revoluções. O computador, por si só, não é um agente de mudanças. Se para o professor, ensinar é transmitir conhecimento, é fixar regras, o computador, com todos os seus recursos de multimídia (som, imagem, animação), será apenas uma versão moderna da máquina de ensinar skineriana.

O que acontece de fato, na grande maioria das vezes, é uma modernização conservadora, onde o "espírito" revolucionário do uso do computador é subvertido pelo sistema educacional vigente e convertido em instrumento de sua consolidação.

Serpa (1986, pg.43) invoca os estudos realizados por Chambers e Sprecher em 1980, mostrando que a informática pode ser empregada em benefício da atividade escolar e acadêmica. Em relação ao ensino tradicional, os estudos concluíram que o ensino assistido pelo computador:

- melhorou a aprendizagem, ou pelo menos, não piorou;
- reduziu o tempo de aprendizagem;
- melhorou as atividades dos estudantes para com o computador no processo de ensino aprendizagem.

Mais recentemente, com um maior emprego da multimídia interativa, Chaves (1991, pg.57) constatou os seguintes benefícios da informática na educação:

- maior motivação e interesse dos alunos;
- ritmo individualizado de aprendizado;
- aumento da quantidade de material aprendido;
- aumento do tempo de retenção do aprendizado;
- redução do tempo de aprendizado

Conclusões semelhantes foram apresentadas por Reinhardat (1995), quando ele afirma que o emprego dos recursos oferecidos pela informática podem:

- aumentar a taxa de retenção dos conhecimentos adquiridos e colaborar com a melhor qualidade do rendimento escolar;

- reduzir o tédio e, em conseqüência, os casos de mau comportamento dos alunos;
- apoiar uma seqüência progressiva de exercícios práticos, individualizados ou em projetos específicos;

Mendes (1995) também relata algumas características e alcance educativo da informática na escola:

- os computadores podem auxiliar o aluno a executar e elaborar tarefas de acordo com seu nível de interesse e desenvolvimento intelectual;
- jogos e linguagens podem auxiliar no aprendizado de conceitos abstratos;
- recurso pode organizar e metodizar o trabalho, gerando uma melhor qualidade de rendimento;
- destaca o elemento afetivo, já que o aspecto motivacional é inerente à relação do aluno com o microcomputador.

Com isto, novos campos de atuação e desafio se abrem para o professor, como aprender a lidar com as novas tecnologias e a integrá-las ao desenvolvimento pleno do ser humano, na área sensorial, emocional e intelectual. Assim, mais do que transmitir conteúdos, o professor vai ensinar o aluno a pensar. Não basta que ele seja um grande especialista (Severino, 1996), ele é "um educador inserido numa situação histórica e cultural(...) que deve saber conduzir os alunos a descobrirem as vias de aprendizagem".

Muitos ainda acreditam que a computação irá criar uma revolução na educação, o que não está acontecendo, como pode-se constatar. As chances de mudar apenas por possuir computadores nas escolas, é quase nula. O que está em foco, e precisa ser entendido é a utilização que vai ser dada à máquina.

O computador pode ser utilizado de várias formas e para diversos fins, dependendo da concepção de educação adotada. Pode ser usado para desenvolver a socialização ou o individualismo, a cooperação ou a competição. Pode ser usado também para desenvolver as estruturas de pensamento ou

para transmitir conhecimentos. É apenas mais um recurso pedagógico cujo uso vai depender da concepção de educação e dos objetivos do professor.

O emprego do computador no processo pedagógico, assim como o uso de qualquer tecnologia exige do educador uma reflexão sobre o valor pedagógico da informática sobre as transformações da escola e sobre o futuro da educação.

Segundo Lina Garcia Cabrera, Rafael M. G. Cabrera e M. L. R. Cejudo em artigo publicado na revista Novatica nº 117 (1995, p.48):

> "Os sistemas educacionais não podem perpetuar-se em seus modelos e métodos de ensino convencionais e, a formação e o uso da tecnologia deve estar presente na grade curricular básica, afim de preparar as novas gerações. O computador no ensino pode assumir tantas e diversas facetas que podemos afirmar que é o recurso didático mais versátil do momento e que é capaz de aglutinar e integrar qualquer tipo de comunicação. O computador nos ambientes educacionais pode ser uma poderosa ferramenta para a gestão de centro de ensino e do grupo bem como a orientação do aluno. Pode ainda ser uma ferramenta de ensino-aprendizagem, tanto para adquirir conhecimentos concretos, assim como meio de trabalho para a pesquisa e desenvolvimento do corpo discente."

O computador pode ser somente um transmissor de informação, ora um treinador, mas também pode fomentar o conhecimento lógico e reflexivo, a imaginação e a criatividade e, inclusive, auxiliar na auto-estima e no desenvolvimento dos esquemas mentais do aluno.

> "Nem todos os alunos tem porque adquirir conhecimentos e habilidades com um único método e com a mesma quantidade de tempo (planificação temporal). A maneira de captar a atenção do aluno depende muito de suas preferências, atitudes e, consequentemente, de seu ambiente familiar e social." [Cabrera, Cabrera e Cejudo, 1995, p.50]

Conseguir não é rápido e nem fácil e, quem sabe, deve-se passar por um período de adequação em que convivam o método de ensino tradicional aliado ou suavizado por ferramentas informáticas sem uma comunicação especial entre ambos. Somente a experimentação fornecerá a linha a seguir para atingir

uma perfeita simbiose e transformação para alcançar um ensino de maior qualidade, cheio de possibilidades, flexível e personalizado.

> "Contamos com múltiplas ferramentas que resolvem um problema concreto e que podem ser aproveitados pelo professor para um determinado objetivo ou uma determinada unidade de uma disciplina. O caminho correto é proporcionar aos professores *softwares* totalmente abertos num duplo sentido: que admitam modificações e adaptações e que permitam incorporar e integrar outras ferramentas educativas." [Cabrera, Cabrera e Cejudo, 1995, p.49]

Pode-se ter a certeza de que o computador não é a panacéia que irá resolver os problemas da educação. O computador em si é neutro, depende de como iremos usá-lo.

A tecnologia pode ajudar a escola levar os seus alunos a um novo paradigma, desde que a tecnologia esteja a serviço das metas educacionais e não a serviço da criação de novas necessidades artificiais. Para Dockterman (1991), a escola ideal é aquela onde a aprendizagem ocorre com e não a partir dos computadores.

A informática quanto adotada nas escolas deve integrar-se ao currículo, não como uma disciplina, mas como uma ferramenta, inclusive, multidisciplinar, constituindo-se em alguma coisa a mais que o professor pode contar para bem realizar o seu trabalho; desenvolvendo atividades que levem a uma reflexão sobre qual a melhor forma de empregar seus recursos, analisando as características de cada disciplina; realizando a imprescindível interação entre as diversas disciplinas e os recursos da informática.

Quando utilizada desta maneira na escola, ou seja, a informática a serviço de um projeto educacional, propicia condições aos alunos de trabalharem a partir de temas, projetos ou atividades, surgidos no contexto da sala de aula. Em decorrência dessas situações os alunos podem contar com a interatividade e a programabilidade possibilitada pelo computador.

Assim sendo, a preocupação fundamental é com o desenvolvimento de valores, com a concepção que temos das finalidades da educação e da convicção de que necessitamos formar um indivíduo com a inteligência desenvolvida, com cultura, flexível, crítico e criativo. A informática pode fazer parte desse universo, mas não pode ser encarada como um objetivo por si própria.

2.4 - Programas de Computador

2.4.1 - Conceito

Para que um computador possa desempenhar uma tarefa é necessário que esta tarefa seja detalhada passo a passo, numa forma compreensível pela máquina, utilizando aquilo que se chama de programa. Neste sentido, um programa de computador nada mais é que um algoritmo escrito numa forma compreensível pelo computador.

Um programa é um conjunto de instruções que dizem ao computador o que fazer. Esse conjunto de instruções, escritas originalmente pelo programador é chamado de código-fonte.

De acordo com a Lei Nº 9.609, de 19 de Fevereiro de 1998, artigo 1º, que dispõe sobre a proteção de propriedade intelectual de programa de computador, comercialização no País, entre outros define:

> "Programa de computador é a expressão de um conjunto organizado de instruções em linguagem natural ou codificada, contida em suporte físico de qualquer natureza, de emprego necessário em máquinas automáticas de tratamento da informação, dispositivos, instrumentos ou equipamentos periféricos, baseados em técnica digital ou análoga, para fazê-los funcionar de modo e para fins determinados."

2.4.2 - Linguagem de Programação

A linguagem de programação é uma forma codificada para escrever as instruções que possibilitam o entendimento pelo computador. Podemos dividir, genericamente, as linguagens de programação em dois grandes grupos: as linguagens de baixo nível (de máquina ou absoluta) e as de alto nível (automática).

As linguagens de baixo nível são voltadas para a máquina, isto é, são escritas usando as instruções do microprocessador do computador. São genericamente chamadas de linguagens Assembly. A figura 2.1 mostra um trecho de programa Assembly. Dentre as vantagens deste tipo de linguagem pode-se destacar a maior velocidade de processamento e o reduzido espaço em memória que ocupam. Quanto as desvantagens, em geral, programas em Assembly tem pouca portabilidade, isto é, um código gerado para um tipo de processador não serve para outro. Códigos Assembly não são estruturados[1], tornando a programação mais difícil.

```
a 100
mov ax, 0002        Move o valor 0002 para o registrador ax
mov bx, 0004        Move o valor 0004 para o registrador bx
add ax, bx          Adiciona o conteúdo dos registradores ax e bx,
                    guardando o resultado em ax
nop
```

Figura 2.1: Exemplo de um programa escrito na linguagem Assembly.

As linguagens de alto nível são voltadas para o ser humano. Em geral utilizam sintaxe estruturada tornando seu código mais legível.

A linguagem de alto nível é aquela que permite, normalmente, escrever o problema em uma linguagem bem próxima àquela em que foi formulado. A criação de linguagens de alto nível foi um passo decisivo para maior facilidade de "comunicação" entro o usuário e o computador.

[1] Não utilizam controles de seleção ou repetição para desvios de execução.

As linguagens de programação necessitam de compiladores ou interpretadores para gerar instruções do microprocessador. Interpretadores fazem a interpretação de cada instrução do programa fonte executando-a dentro de um ambiente de programação, Basic e AutoLISP por exemplo. Compiladores fazem a tradução de todas as instruções do programa fonte gerando um programa executável. Estes programas executáveis (*.exe) podem ser executados fora dos ambientes de programação, C e Pascal por exemplo. As linguagens de alto nível podem se distinguir quanto a sua aplicação em genéricas como C, Pascal, Delphi, Visual Basic ou específicas como Fortran (cálculo matemático), GPSS (simulação), LISP (inteligência artificial). Existe apenas uma linguagem que pode ser compilada e interpretada, a Linguagem Forth, pois ela tem comandos que são compilados e outros que são interpretados.

```
#include <stdio.h>
void main(){
        unsigned int i,n,fat;
        do{                                     // leitura do numero
                puts("\nDigite um inteiro entre 0 e 10: ");
                scanf("%u",&n);
        }while(n < 0 || n > 10);                // repete a leitura se n < 0 e n > 10
        fat = 1;
        i = 1;                                  // contador de iteracoes
        do{                                     // calculo do fatorial
                fat *= i;                       // fat = fat * i
        }while(i++ <= n);                       // repete multiplicacao se i < n
        printf("\n %u! = %u",n,fat);
}
```

Figura 2.2: Exemplo de um programa escrito na linguagem C.

As vantagens das linguagens por serem compiladas ou interpretadas, tem maior portabilidade podendo ser executados em varias plataformas com pouquíssimas modificações. Em geral, a programação torna-se mais fácil por causa do maior ou menor grau de estruturação de suas linguagens. Entretanto, em geral, as rotinas geradas (em linguagem de máquina) são mais genéricas e portanto mais complexas e por isso são mais lentas e ocupam mais memória.

2.4.3 - A Evolução

No início do século XIX, Joseph Marie Jacquard (1752-1834) construiu um tear automático que utilizava cartões perfurados para controlar a confecção dos tecidos e seus desenhos. Podemos considerar a primeira máquina mecânica programada.

Embora as calculadoras mecânicas tenham sido introduzidas no século XVII por Pascal e Leibniz, o trabalho de Charles Babbage (1792-1871), matemático inglês foi mais complexo. Ele criou, em 1842, um mecanismo que utilizava cartões perfurados para armazenar e recuperar informações. O trabalho de Babbage constituiu o fundamento para os computadores modernos.

Um passo para a computação automatizada foi a introdução de cartões perfurados, dos quais seu primeiro uso bem-sucedido em conexão com a computação foi em 1890 por Herman Hollerith e James Powers que trabalhavam para a Agência de Censo norte-americana. Eles desenvolveram dispositivos que poderiam ler a informação que tinha sido perfurada em cartões automaticamente, sem o intermédio humano.

Interesses comerciais logo conduziram ao desenvolvimento de sistemas de máquinas perfuradoras de cartões como a International Business Machines (IBM) , Remington-Rand, Burroughs, e outras corporações.

Quando os cientistas conseguiram substituir os desajeitados e pesados mecanismos de engrenagens e alavancas pela corrente elétrica, os avanços no desenvolvimento dos computadores e, consequentemente, dos programas foram rápidos.

Em 1937, as técnicas das máquinas perfuradoras de cartão tinham se tornado tão bem estabelecidas e confiáveis que Howard Hathaway Aiken, em colaboração com engenheiros da IBM, empreendeu a construção de um grande computador digital automático baseado no padrão IBM das partes eletromecânicas. O MARK-I, como foi chamado, e concluído em 1944, possuía

unidades de entrada, memória principal e unidade de saída. Grace Murray Hopper foi uma figura muito importante na evolução dos computadores e no desenvolvimento das linguagens de programação. Inclusive, foi responsável por encontrar o 1° *bug* no sistema do MARK-I, causando por uma mariposa morta em contato com os reles.

Em 1942, John P. Eckert, John W. Mauchly e seus sócios, na Escola Moore de Engenharia Elétrica da Universidade da Pennsylvania decidiram construir um computador eletrônico de alta velocidade para realizar o trabalho. Esta máquina foi conhecida como ENIAC - "Integrador e Calculador Numérico Elétrico" (Electrical Numerical Integrator And Calculator). Consistia principalmente em válvulas e relês e tinham de ser programados conectando-se uma série de plugues e fios. Uma equipe de programadores podia passar vários dias introduzindo um pequeno programa em uma dessas máquinas que ocupavam salas inteiras.

Intrigado pelo sucesso do ENIAC, o matemático John Von Neumann empreendeu em 1945 um estudo teórico de computação que demonstrou que um computador poderia ter uma estrutura física muito simples e fixa e que ainda poderia ser capaz de executar qualquer tipo de cálculo efetivamente por meio de controle programado formal sem a necessidade por qualquer mudança em *hardware*. Von Neumann contribuiu com uma nova compreensão de como deveriam ser organizados e construídos computadores rápidos e práticos. Tornaram-se fundamentais para gerações futuras de computadores de alta velocidade e que foram adotadas universalmente. Como resultado, programar ficou mais rápido, mais flexível, e mais eficiente.

Em 1951, Mauchly constrói o UNIVAC-I (Computador Automático Universal), que utilizava fitas magnéticas.

Nos anos sessenta, grandes avanços em aplicações que programam linguagens remove muitos obstáculos. Linguagens de aplicações estão agora disponíveis para controlar um grande número de processos computacionais.

Em 1964, Thomas Kurtz e John Kemeny, professores do DartMouth College, criaram o BASIC, uma linguagem de programação de fácil aprendizagem.

Em 1966, a Hewlett-Packard entrou no negócio de computadores para uso geral com seu HP-2115. Ele suportava uma grande variedade de linguagens, entre elas BASIC, ALGOL e FORTRAN.

Em 1967, Seymour Papert desenhou o LOGO como se fosse uma linguagem de computação para crianças. Inicialmente como um programa de desenho, o LOGO controlava as ações de uma "tartaruga" mecânica, que traçava sua trilha com caneta em um papel.

Em 1969, programadores dos laboratórios AT&T Bell, Ken Thompson e Denis Richie desenvolvem o UNIX, primeiro sistema operacional que poderia ser aplicado em qualquer máquina. Ainda neste ano, o exército americano interligou as máquinas da Arpanet, formando a rede que originaria a Internet.

Em 1970, o SRI Shakey foi o primeiro robô móvel internacional controlado por inteligência artificial. E é a partir da inteligência artificial que 5 anos mais tarde criou-se um programa chamado MYCIN, um sistema especialista que ultrapassa em precisão a capacidade médica de diagnosticar meningite em pacientes. Edward Feigenbaum, da Universidade de Stanford, pioneiro na tecnologia de sistemas especialistas define-os com "...um programa inteligente de computador que usa conhecimento e procedimentos de inferência para resolver problemas que são difíceis o suficiente para que sua solução necessite de um grau significativo de perícia humana.". Isto é, um SE é um sistema de computação que emula a habilidade de tomar decisões de um especialista humano.

A partir daí, o progresso na área de *software* não se emparelhou aos passos largos em *hardware*. O *software* se tornou o custo principal de muitos sistemas porque a produtividade em programação não aumentou muito rapidamente. Foram desenvolvidas técnicas de programação novas, como programação orientada à objeto, para ajudar a aliviar este problema. Apesar das dificuldades

com o *software*, porém, está diminuindo o custo por cálculo de computadores rapidamente, e sua conveniência e eficiência deve aumentar num futuro previsível.

O campo dos computadores continua experimentando tremendo crescimento. Redes de computador, correio eletrônico (*e-mail*) e publicações eletrônicas são apenas algumas das aplicações que amadureceram nos últimos anos.

2.5 - O Processo Pedagógico para o Ensino da Lógica de Programação

2.5.1 - A Prática

A abordagem do conteúdo de disciplinas como lógica ou técnicas de programação tem como principal objetivo o aprendizado para o desenvolvimento de programação em computadores que consiste de aspectos pedagógicos, em sua maioria, desconhecidos para os alunos.

> "Pode-se dizer que, no nível de estruturas mentais, a mudança de desenvolvimento ocorre em determinados passos, de uma maneira que pode ser associada com a subida de uma escada, em oposição à subida de uma ladeira íngreme. De acordo com alguns psicólogos, as aquisições do aluno são limitadas pelo estágio que ela alcançou: se uma habilidade específica exige que um aluno esteja num estágio mais avançado, não adianta tentar ensinar tal técnica, até que o estágio necessário seja alcançado. Além disso, os psicólogos da linha de Piaget argumentam que o avanço de um estágio para outro não pode ser conseguido através de exemplos específicos de aprendizagem: uma variedade de experiências é necessária." [HOWE, 1986, p.98]

Normalmente, a disciplina é contemplada por uma expectativa e ansiedade na qual a maioria dos alunos deseja dominar. Descobrir o processo de desenvolvimento de aplicações é o ponto onde estudantes de informática priorizam como conhecimento formador de suas atividades.

O processo pedagógico tende a ser uma atividade imprescindível no sentido de proporcionar aos alunos uma assimilação e conseqüente acomodação de técnicas de raciocínio lógico que fundamentarão desempenhos na resolução de

algoritmos de complexidade diversas e consequentemente de futuros programas.

Inicialmente, busca-se discriminar o processo de um programa através de alguma atividade cotidiana. Ressaltando o fato de que cada passo deve ser especificado detalhadamente, além de obedecer uma ordem seqüencialmente lógica. A forma, naturalmente, utilizada para descrever cada passo é a descrição narrativa. Esta forma permite o desenvolvimento de algoritmos sem nenhuma regra específica determinada. É como uma receita. Esta fase começa a determinar conceitos lógicos que serão fundamentais para um bom programador. Começa neste estágio a formação das estruturas cognitivas através dos esquemas para futuras reutilizações. No entanto, não será desta forma que os programas serão gerados. Há um bom caminho pela frente. À cada fase um novo degrau. É o processo de construção do conhecimento.

Segundo Howe (1986), o conhecimento já existente é, talvez, o fator mais importante: as pessoas utilizam o que elas já sabem, com o objetivo de compreender novas informações. Tal conhecimento pode ser em forma de dados, sobre as qualidades de um item ou acontecimento, ou pode ser grandes esquemas contendo informação organizada, sobre uma seqüência de acontecimentos. Por exemplo: para ir a algum lugar, elaborar o trajeto a ser seguido; para utilizar uma variável em um programa, declarar esta variável.

Num conteúdo que propõem a utilização do computador para desenvolver programas, a expectativa tende a aumentar pois um conhecimento mais organizado e experiências de situações variadas devem permear o aluno possibilitando ao mesmo o desenvolvimento lógico de algoritmos.

Nestas atividades de ensino utiliza-se técnicas que permitem o desenvolvimento de algoritmos através de uma forma mais próxima de alguma linguagem de programação. Normalmente, a linguagem que será posteriormente utilizada. A técnica é conhecida como pseudocódigo. O pseudocódigo é um processo formal para desenvolvimento de algoritmos numa linguagem mais próxima de seu conhecimento. Em nosso caso: o português.

Por este motivo, esta técnica também é conhecida como: Portugol. O Pseudocódigo é o processo mais explorado por ser o mais importante no processo de aprendizagem da lógica de programação.

Esta técnica é empregada com dois objetivos:

1- Construção do conhecimento lógico formalizado e regido (não rigorosamente) por regras quanto a forma e estrutura que um algoritmo deve ter e;
2- Adaptação para posterior emprego do conhecimento em linguagens de programação.

Para Saliba (1992), esta forma de representação é rica em detalhes por assemelhar-se bastante à forma em que os programas são escritos. Ela é suficientemente geral para permitir que a tradução de um algoritmo nela representado para uma linguagem de programação específica seja praticamente direta.

Por exemplo, na figura abaixo, temos um trecho de um algoritmo que lê dois valores, atribui a adição desses dois valores em uma variável e depois apresenta este resultado.

```
...
LEIA VALOR1;
LEIA VALOR2;
SOMA <- VALOR1 + VALOR2;
ESCREVA SOMA;
...
```

Figura 2.3: Exemplo de algoritmo em pseudocódigo.

É comum, paralelo ao pseudocódigo, a utilização da técnica de fluxogramas. Fluxograma é uma representação gráfica de algoritmos onde formas geométricas diferentes implicam ações (instruções, comandos) distintos. Tal propriedade facilita o entendimento das idéias contidas nos algoritmos e justifica sua popularidade. Há formas geométricas apropriadas para representar cada um dos diversos tipos de instruções. A figura 2.4 mostra o algoritmo da figura 2.3 em forma de fluxograma.

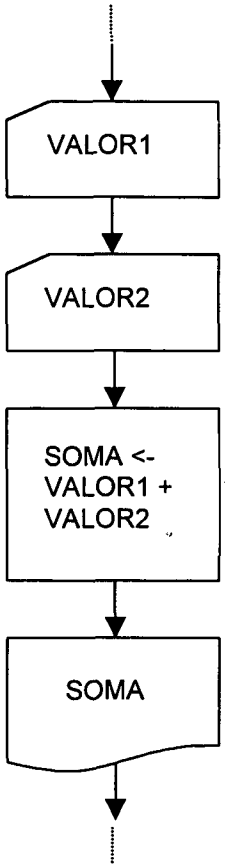


Figura 2.4: Exemplo de um fluxograma.

Após o extenso embasamento teórico proporcionado para o desenvolvimento de algoritmos atribuindo neste ponto uma considerável experiência de raciocínio lógico necessário para tal desenvolvimento, os alunos tem, consequentemente, o pré-requisito básico para utilização de uma linguagem de programação.

Torna-se neste momento, o tão esperado uso de uma ferramenta que transferirá os conhecimentos abstratos - até então somente realizados numa "folha de papel" - para um ambiente que proporciona a verificação (normalmente, através da compilação) do programa, permitindo aplicação prática do conhecimento. Após a fase de compilação, avalia-se a consistência lógica do programa desenvolvido através de sua execução, isto é, se o programa realiza o que foi previsto.

A praticidade gerada pelos ambientes de programação proporciona a acomodação do conhecimento nas estruturas cognitivas permitindo que técnicas aplicadas em situações passadas possam ser resgatadas para resolver problemas posteriores.

De acordo com Howe (1986), a extensão das técnicas e habilidades adquiridas do indivíduo dependerão de vários fatores que influenciam a aprendizagem de uma maneira ou de outra. Estes incluem, em adição às influências motivadoras e diferenças na atenção e no uso de estratégias de aprendizagem eficazes, determinantes adicionais da aprendizagem, tais como diferenças individuais na percepção, perseverança, impulsividade e um número de variáveis de temperamento, o conhecimento já existente da pessoa e técnicas. Partindo do ponto de que este último aspecto tem uma influência cumulativa sobre as aquisições aprendidas do indivíduo, segue que a aprendizagem prévia afetará o grau de sucesso da pessoa, em novas tarefas de aprendizagem.

2.5.2 - Os Problemas

Como todo processo de ensino-aprendizagem, o professor deve estar munido de técnicas que viabilize e contemple o aprendizado. E de acordo com estas técnicas, pode atingir um maior ou menor grau de produtividade neste processo.

Mas, cada pessoa possui métodos e formas diferentes de aprender. Além de vários outros fatores, a observação de Gardner contribuiu para pedagogos oportunizarem outras formas com o objetivo de promover meios alternativos "mais adequados". Este fator implica no aperfeiçoamento qualitativo de aprendizado estabelecido através de técnicas que proporcionassem maior atenção.

Cada pessoa apresenta uma forma, um modo particular singular de entrar em contato com o conhecimento. Isto quer dizer que cada um tem uma particular e individual modalidade de aprendizagem, que oferece uma maneira própria de se aproximar do objeto de conhecimento, formando um saber que é peculiar.

No entanto, a forma tradicional no processo ensino-aprendizagem da lógica de programação, consiste principalmente na assimilação através de aulas expositivas onde o aluno tem grandes dificuldades de acomodar o conhecimento, haja visto que, a resposta de um problema que é um conjunto de supostas instruções acaba sendo uma incógnita. De forma autônoma é difícil para um aluno avaliar se o algoritmo está escrito corretamente (sintática e semanticamente), e se o resultado lógico é o esperado.

A elaboração de algoritmos consiste num atividade que, para a maioria dos alunos, é totalmente nova. A técnica da Descrição Narrativa aplicado no processo de ensino-aprendizagem da lógica de programação inicia o processo de construção das estruturas cognitivas para o desenvolvimento formal de algoritmos.

Outra dificuldade é a não existência de ferramenta adequada que proporcione uma representação cognitiva genérica, ou seja, todos os resultados são validados, normalmente, através da correção pelo professor. Neste caso, abstrai-se o funcionamento do algoritmo simulando-o através do que chamamos de "teste de mesa" a execução das instruções. A abstração do conhecimento acaba sendo o único aliado no processo de aprendizagem. Esta abstração na resolução de problemas consiste numa das qualidades que poucos alunos desenvolvem.

Como o objetivo é preparar os alunos para desenvolver programas eficientes e logicamente corretos, é de fundamental importância a compreensão de cada instrução que irá compor um programa, bem como a relação que deverá existir entre estas instruções.

A saída encontrada nestas situações acaba sendo, obviamente, a utilização de linguagens de programação para testar e concretizar o aprendizado. Todavia, os alunos que ainda não assimilaram o processo de construção de algoritmos terão, consequentemente, dificuldades iniciais maiores para a aprendizagem.

As linguagens de programação normalmente desenvolvidas para propósitos específicos como desenvolvimento de aplicações comerciais ou para finalidades científicas, não atendem a critérios que seriam benéficos para pessoas que estão iniciando.

A interface do ambiente de programação, a gramática da linguagem, as mensagens de erros acabam sendo novos problemas. Poderíamos considerar, também, a falta ou a clareza de um sistema apropriado de ajuda. O ambiente, normalmente, não é apropriado, ou pela complexidade da linguagem ou pela falta de interatividade com o usuário.

A gramática é, normalmente, inconsistente se comparado a forma como foi aprendida utilizando o pseudocódigo. Isto acaba gerando conflitos e muitas vezes a desmotivação do aluno para aprender. Entre outros fatores, a desmotivação é uma das grandes causas de desistências e reprovações nestas disciplinas.

Alia-se a este fato o problema que é acarretado pela falta de um tratamento de erros de programas mais adequado. Desconsiderando o fato de que as mensagens são em "inglês codificado", elas são em sua maioria incompletas e acabam gerando dúvidas ainda maiores. A insegurança está permanentemente presente na fase de elaboração de programas porque não foi explorado de maneira eficiente a questão cognitiva na aquisição de conhecimento para elaboração de programas. Uma vez assimilado esse conhecimento num ambiente adequado, o processo seguinte que seria a utilização de linguagens de programação tornar-se-ia mais facilitado.

> "Os programadores teriam seu trabalho facilitado se os programas fossem escritos em sentenças padronizadas em linguagem humana; infelizmente isso não acontece." [QUE, 1993, p.5]

Morente (1966, p.27) coloca que:

> "Um dos problemas que a teoria do conhecimento tem de propor e solucionar é saber quais os critérios, as maneiras e os

> métodos de que se pode valer o homem para ver se um conhecimento é ou não verdadeiro."
>
> "Os problemas variarão de acordo com a extensão em que vestígios reativados de operações passadas serão suficientes para fornecer uma solução ou, alternativamente, se será necessária uma extensa reestruturação simbólica da experiência anterior." [Greene, 1975, p.26]

É fundamental o desenvolvimento de metodologias alternativas que busquem a perfeita integração no processo ensino-aprendizagem. O que se percebe é que a aprendizagem da lógica de programação é um salto sobre um abismo: alguns conseguem salta-lo sem grandes dificuldades; outros, quase caem, se agarram na sua beira e com dificuldades conseguem; outros, por maior esforço que façam não conseguem o almejado "outro lado"; e outros, o que é pior, nem se quer arriscam. É necessário criar alternativas para possibilitar a passagem por esse "abismo".

2.6 - A Linguagem Logo

Pode-se dizer que a linguagem Logo teve sua origem no final dos anos 60, no Instituto de Tecnologia de Massachusetts, nos Estados Unidos. Foi desenvolvida por um grupo de pesquisadores, liderados pelo matemático e educador Seymor Papert, considerado o "pai do Logo". O ambiente Logo reflete uma Teoria da Educação, tendo como centro a criança e sua forma de pensar e agir no mundo. (Valente, 1996)

O Logo, desde sua criação e até 1976 ficou restrito a estudos e aplicações de laboratórios, tais como: o MIT (Instituto de Tecnologia de Massachusetts), o Departamento de Inteligência Artificial da Universidade de Edinburgh e Instituto de Educação da Universidade de Londres.

A preocupação desses pesquisadores concentrava-se principalmente em desenvolver *hardware* e *software* para implementar o Interpretador Logo e em demonstrar o que se podia fazer com ele, tendo em vista principalmente sua relação com a matemática.

Trabalhava-se com ele em computadores de médio e grande porte, o que também foi um dos fatores contribuintes para que seu uso de início, ficasse restrito à universidades e centros de pesquisa.

O Logo nasceu com base nas referências teóricas sobre a natureza da aprendizagem desenvolvidas por Piaget (reinterpretadas por Papert), e nas teorias computacionais, principalmente a da Inteligência Artificial, vista como Ciência da Cognição, que para Papert também é uma metodologia de ensino-aprendizagem, cujo objetivo é fazer com que as crianças pensem a respeito de si mesmas.

O projeto, começado em 1977 na Escola Pública de Brookline, usando Logo em um micro-computador 3500 (criado por Marvin Minsky), aplicado em 16 alunos da 6ª série, começou a sair dos laboratórios e penetrar na escola.

A princípio não houve preocupação com o papel do professor no ambiente Logo. Papert no livro Logo: Computadores e Educação (1985), traduzido da obra publicada em 1980 nos Estados Unidos sob o título de "Mindstorms: Children, Computers and Powerful Ideas", deixa transparecer que "idéias poderosas" surgiriam espontaneamente da atividade do aluno ao programar em Logo e isso aconteceria sem uma maior intervenção do professor, cabendo-lhe apenas auxiliar os alunos no que diz respeito à sintaxe do Logo.

Logo não é só o nome de um linguagem de programação, mas também de uma filosofia que lhe é subjacente. A filosofia surgiu dos contatos de Papert com a obra de Piaget e dos estudos sobre o problema da inteligência artificial.

A visão que Papert tem do homem e do mundo situa-se numa perspectiva interacionista, sendo o conhecimento o produto dessa interação, que é centrada nas formas com que o mundo cultural age e influencia o sujeito em interação com o objeto. Ao contrário de Piaget, Papert enfatiza que aquilo que aprendemos e o como aprendemos depende dos materiais culturais que encontramos à nossa disposição.

De acordo com sua teoria do conhecimento e do desenvolvimento humano, o processo educacional tem como pressuposto que a criança não aprende apenas pelo ensino formal e deliberado, que ela é uma aprendiz inata, que mesmo antes de chegar à escola apresenta conhecimentos adquiridos por meio de uma aprendizagem natural, espontânea e intuitiva, que se dá através da exploração, da busca e da investigação, a qual pode ser caracterizada como uma real auto-aprendizagem.

Aquilo que a criança aprendeu porque fez, após ter explorado, investigado e descoberto por si própria, além de contribuir para o desenvolvimento de suas estruturas cognitivas, reveste-se de um significado especial que ajuda a reter e transferir com muito mais facilidade aquilo foi aprendido.

Está imbuída na filosofia do Logo, como a concebeu Papert, a idéia que a aquisição de um conhecimento não se dá em função do desenvolvimento, mas principalmente na maneira pela qual as pessoas se relacionam com o meio, ou seja, as condições que este oferece para exercitar o pensamento qualitativo. Acredita na necessidade da pessoa controlar sua aprendizagem, poder reconhecer e escolher entre várias possibilidades de pensamento estruturado.

Um outro aspecto importante nas concepções de Papert, é o fato de no Logo considerar-se o erro como um importante fator de aprendizagem, o que oferece oportunidades para que o aluno entenda porque errou e busque uma nova solução para o problema, investigando, explorando, descobrindo por si próprio, ou seja, a aprendizagem pela descoberta.

Os procedimentos de análise e correção no processo de aprendizagem pelo Logo possibilitam a descoberta de diferentes caminhos na solução de problemas, sendo que esses caminhos advém de um contexto cultural onde não há certo e errado pois as soluções são pessoais.

É esse tipo de aprendizagem que a filosofia do ambiente Logo pretende que seja desenvolvida com a ajuda da linguagem de programação Logo, que possibilita integrar habilidades corporais com as intelectuais, a visualização da

representação do modo como pensamos, promovendo o desenvolvimento do pensamento estruturado, modular.

Para ele, o computador pode concretizar e personalizar o formal e sendo bem utilizado permite abordar de forma concreta os conhecimentos até então somente acessíveis via processos formais (Papert, 1985), o que permitiria transpor o obstáculo na passagem do pensamento concreto para o abstrato (identificação de Piaget).

Essa mediação, tanto nos seus aspectos físicos como simbólicos seria feita pela "tartaruga" (cursor gráfico que se tornou símbolo do Logo), mecânica ou de tela, que se move no espaço ou na tela em resposta a comandos fornecidos ao computador.

O Logo é uma linguagem de programação e como tal serve para viabilizar a comunicação com o computador. Essa linguagem possui como todas, seus aspectos computacionais, e no caso do Logo, o aspecto da metodologia para explorar o processo de aprendizagem.

O importante de todo esse processo é que a criança vai aprendendo, de acordo com os pesquisadores, conceitos e princípios importantes, não só de geometria, mas também sobre como resolver um problema.

O Logo propõe um ambiente de aprendizagem no qual o conhecimento não é meramente passado para o aluno, mas, uma forma de trabalho onde esse aluno em interação com os objetos desse ambiente, possa desenvolver outros conhecimentos, por exemplo: conceitos geométricos ou matemáticos. Propicia ao aluno a possibilidade de aprender fazendo, ou seja, ensinando a tartaruga a resolver um problema, seguindo a linguagem de programação. O aluno pode ao ver o resultado da execução, comparar suas expectativas originais com o produto obtido, analisando suas idéias e os conceitos que usou. Se houver um erro o aluno pode depurar o programa e identificar a origem do erro, usando o erro de modo produtivo, para entender melhor suas ações.

A literatura pró-Logo e as abordagens teóricas a respeito de seus pressupostos filosóficos e metodológicos, como os trechos destacados abaixo, nos levam a reflexões sobre a viabilidade da prática efetiva, sob a perspectiva da situação atual de nossas escolas e da formação de nossos professores para tanto.

> "O uso do Logo pode resgatar a aprendizagem construtivista e tentar provocar uma mudança profunda na abordagem do trabalho nas escolas. Uma mudança que coloca a ênfase na aprendizagem ao invés de colocar no ensino; na construção do conhecimento e não na instrução." [Valente, 1996, p.54];
>
> "Primeiro, o controle do processo de aprendizagem, está nas mãos do aprendiz e não nas mãos do professor. Isto porque a criança tem a chance de explorar o objeto 'computador' de sua maneira e não de uma maneira já pré-estabelecida pelo professor. É a criança que propõe os problemas ou projetos a serem desenvolvidos através do Logo..." [Valente, 1996, p.49]

A filosofia construtivista que dá suporte aos pressupostos educacionais do Logo, e não apenas dele, pois trata-se de um dos paradigmas das idéias pedagógicas, ou seja, o paradigma que vê a educação com um processo de desenvolvimento do ser humano, o qual, em condições adequadas obtém o afloramento das suas reais potencialidades, obtendo ao final deste processo, um ser autônomo, que aprende por si próprio, dado que aprendeu a aprender por meio de um processo de investigação, descoberta e invenção, construindo portanto sua própria aprendizagem. Do paradigma centrado no ensino passa-se para o paradigma centrado na aprendizagem.

Essa mudança de paradigma não consegue ser aplicada na maioria das escolas através de outros meios e outras ferramentas já conhecidos, imaginemos então as reais dificuldades para verdadeiramente praticar os pressupostos desse paradigma, quando necessita-se que o professor aprenda e domine uma nova linguagem, tão diferente do material com que vem trabalhando, e ainda aplique aquilo que muitas vezes não consegue fazer nem com o que já domina plenamente.

A criação em geral de ambientes de aprendizagem usando o computador, exige uma formação mais sólida e mais ampla dos professores, tanto no domínio de aspectos computacionais quanto do conteúdo curricular. Não se trata apenas de dominar *hardware* e *software* e sim conhecer profundamente o próprio conteúdo e o modo como o computador pode ser integrado no trabalho com este conteúdo.

CAPÍTULO III

3 - O DESENVOLVIMENTO DE COMPILADORES

Neste capítulo é abordado todo o processo necessário na construção de compiladores. Suas características e finalidades devem estar devidamente planejadas pois a construção de um compilador demanda tempo e mão-de-obra especializada.

3.1 - Introdução

Um compilador pode ser definido de acordo com Aho, Sethi e Ullman (1995, p.1) como:

> "... um programa que lê um programa escrito numa linguagem - a linguagem *fonte* - e o traduz num programa equivalente numa outra linguagem - a linguagem *alvo*. Como importante parte do processo de tradução, o compilador relata ao usuário a presença de erros no programa fonte."

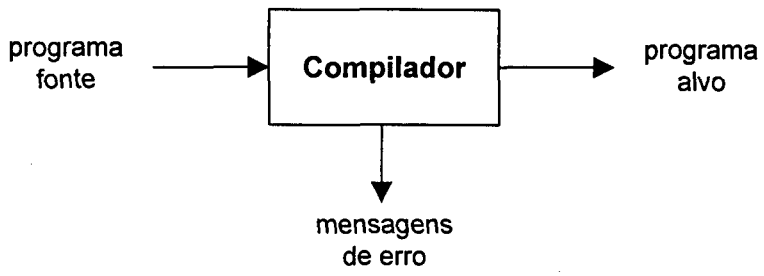


Figura 3.1 – Estrutura Básica de um Compilador.
Fonte: Aho, Sethi e Ullman, p.1.

> "Um compilador é um programa de computador que tem a finalidade de traduzir ou converter um programa escrito em uma linguagem-fonte para um programa escrito em uma outra linguagem-objeto." [Setzer e Melo, 1983, p.12]

Há uma variedade muito grande de compiladores. Existem milhares de linguagens fonte, que vão das linguagens de programas tradicionais, tais como Fortran e Pascal, às linguagens especializadas que emergiram em quase todas as áreas de aplicação de computadores. A despeito da complexidade, as tarefas básicas que qualquer compilador precisa realizar são essencialmente as mesmas.

É difícil fornecer uma data exata para o primeiro compilador. Estima-se que no início dos anos 50. Descobriu-se, desde então, técnicas sistemáticas para o tratamento de muitas das mais importantes tarefas que ocorrem durante a compilação.

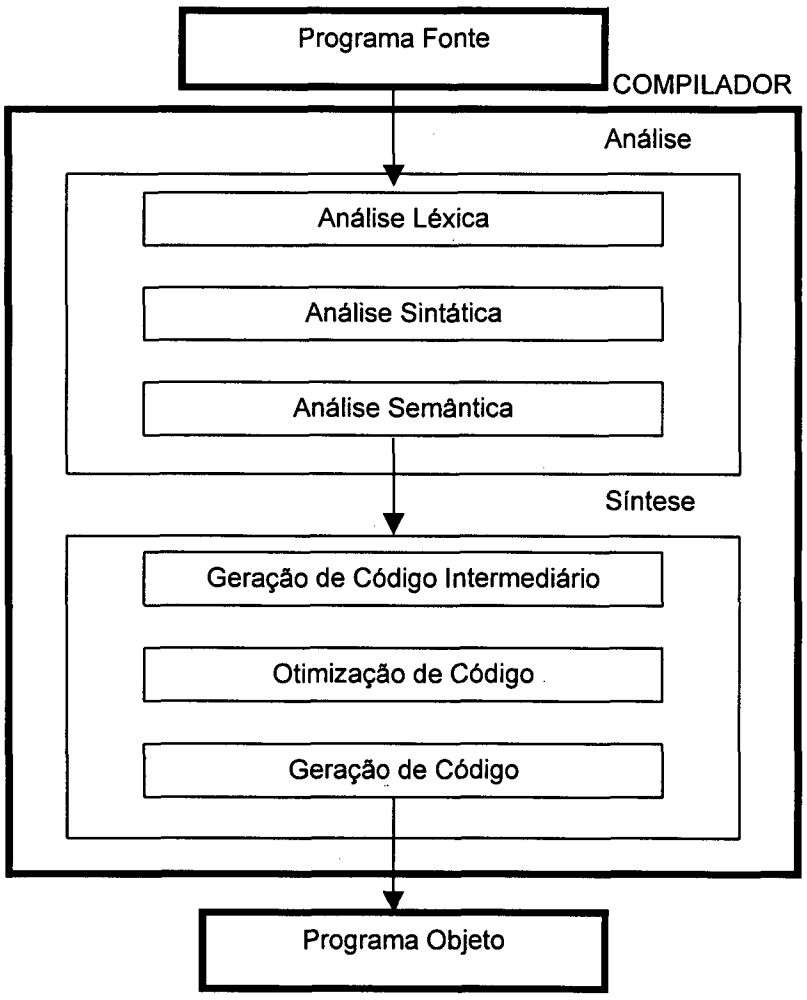


Figura 3.2 – Estrutura Geral de um Compilador.

3.2 - Estrutura Geral de um Compilador

No desenvolvimento do compilador existem duas partes neste processo (figura 3.2): a análise e a síntese. A parte de análise divide o programa fonte nas partes constituintes e cria uma representação intermediária do mesmo. A de síntese constrói o programa objeto desejado a partir da representação intermediária.

3.3 - Formas de Implementação de um Compilador

Quanto ao processo de implementação, o compilador pode ser composto de várias fases, denominadas neste caso de passos do compilador, ou seja, a quantidade de vezes que o programa fonte é analisado até que o código objeto seja gerado.

Segundo Furtado (1987), existem alguns critérios para se definir o número de passos para um compilador:

- memória disponível;
- tempo de compilação;
- tempo de execução;
- características da linguagem;
- características das aplicações;
- disponibilidade de ferramenta de apoio;
- tamanho/experiência da equipe e prazo para desenvolvimento

Setzer e Melo (1983) descrevem vantagens e desvantagens na construção de compiladores com vários passos. As principais vantagens são:

- menor utilização de memória de computadores, pois cada parte exerce apenas uma parte das funções de todo o compilador;
- maior possibilidade de se efetuar otimizações levando a um programa objeto mais eficiente quanto ao espaço ocupado e ao tempo de processamento;
- os projetos e implementações das várias partes do compilador são mais independentes.

As principais desvantagens são:

- maior volume de entrada/saída, quando os programas intermediários e os passos do compilador não ficam residentes na memória;
- normalmente, aumento de tempo de compilação;
- aumento do projeto total.

Um caso interessante foi o compilador Fortran para o IBM-1401 do início de 60. O compilador possuía 64 passos para uma pequena memória de 4 ou 8K.

3.4 - Fases na Construção de um Compilador

3.4.1 - Análise Léxica

O analisador léxico é a primeira fase de um compilador. É a interface entre o programa e o compilador. O programa é lido da esquerda para a direita com o objetivo de agrupar os caracteres em itens léxicos chamados de *tokens* produzindo uma seqüência para a análise sintática. Os *tokens* representam cadeias de caracteres no programa fonte, e podem receber um tratamento conjunto, como instâncias de uma mesma unidade léxica (por exemplo, identificadores, números, etc.).

Segundo Setzer e Melo (1983), na maioria das linguagens de programação, as seguintes construções são tratadas como *tokens*: identificadores, palavras reservadas, constantes numéricas, literais e símbolos especiais.

O analisador léxico ignora "brancos" e comentários, além disso, proporciona detecção e diagnóstico de erros léxicos como por exemplo: símbolos inválidos, literais ou comentários sem fim.

3.4.2 - Análise Sintática

Cada linguagem de programação possui as regras que descrevem a estrutura sintática dos programas. A sintaxe das construções de uma linguagem de

programação pode ser descrita por gramáticas. As gramáticas oferecem uma especificação sintática precisa e fácil de entender.

Segundo Aho, Sethi e Ullman (1995), a função do analisador sintático é obter uma cadeia de *tokens* proveniente do analisador léxico e verificar se a mesma pode ser gerada pela gramática da linguagem-fonte.

Sobre gramática, Furtado (1987, p.11) define como:

> "... uma gramática define uma estrutura sobre um alfabeto de forma a permitir que apenas determinadas combinações sejam válidas, isto é, sejam consideradas sentenças (definindo assim a linguagem que ela representa)."

E acrescenta:

> "... pode ser definida como sendo um dispositivo formal usado para especificar de maneira finita uma linguagem potencialmente infinita."

O processo de análise sintática envolve o agrupamento dos *tokens* do programa fonte em frases gramaticais ou estruturas sintáticas, verificando se a sintaxe da linguagem na qual o programa foi escrito está sendo respeitada, caso contrário, proporciona o diagnóstico de erro sintático.
Por exemplo:

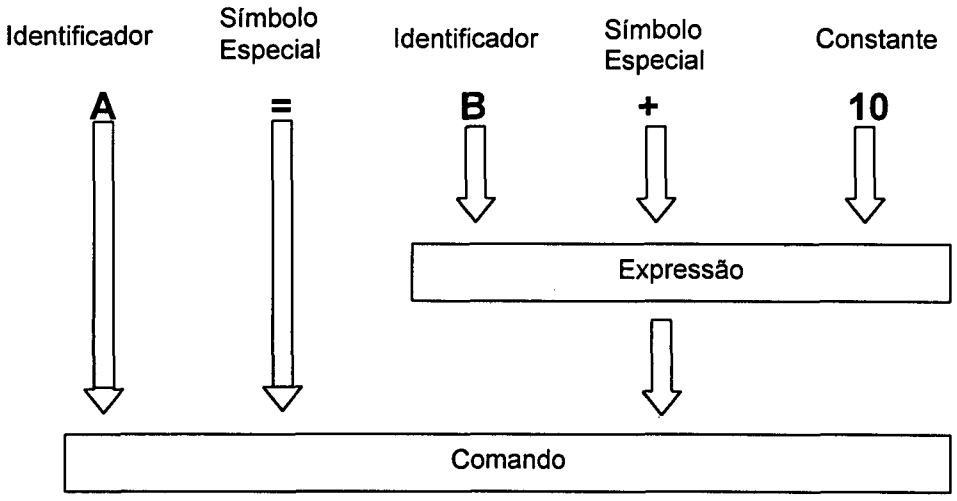


Figura 3.3 – Reconhecimento Sintático de um Comando de Atribuição.

3.4.3 - Análise Semântica

Tem a função de verificar se as construções utilizadas no programa fonte estão semanticamente corretas, isto é, se existe significado ou sentido lógico nas composições das estruturas sintáticas, diagnosticando os erros semânticos. Por exemplo: No comando da figura 3.3 poderá existir ações semânticas que verifiquem se:

- identificador A foi declarado como variável simples;
- identificador B foi declarado como variável simples ou constante;
- identificador B é do tipo numérico para realizar uma adição com a constante 10, ou seja, se a expressão é compatível;
- identificador A é compatível com a expressão.

A fase de análise semântica é o processo inicial onde pode-se viabilizar informações do programa fonte que permitam a geração de código intermediário.

3.4.4 - Geração de Código Intermediário

É uma representação intermediária de instruções equivalentes ao programa fonte para uma máquina hipotética. Esta representação deve possuir duas propriedades importantes: ser fácil de produzir e fácil de traduzir no programa objeto. A representação intermediária pode ter uma variedade de formas.

Exemplo: no comando da figura 3.3 (A = B + 10), o código intermediário gerado, poderia ser algo parecido com:

- CRVL B - Carrega o conteúdo de memória de B
- CRCT 10 - Carrega a constante 10
- SOMA - Soma os dois últimos valores carregados (B e 10)
- ARMZ A - Armazena valor na posição de memória A

3.4.5 - Otimizador de Código

A fase de otimização tenta melhorar o código intermediário de tal forma que venha resultar um código de máquina mais rápido em tempo de execução.

Para Aho, Sethi e Ullman (1995, p.254), o ideal seria que os compiladores produzissem um programa objeto que fosse tão bom quanto aquele que poderia ser escrito manualmente. No entanto, o código intermediário produzido no processo de compilação pode ser transformado de modo a melhorar o código de máquina de forma que a execução seja mais eficiente quanto ao tempo de execução, ao espaço ocupado em memória ou em ambos as situações.

Existe uma grande variação na quantidade de otimizações de código que cada compilador executa. Naqueles que mais a realizam, chamamos de "compiladores otimizantes", uma porção significativa é gasta nesta fase.

3.4.6 - Gerador de Código

A fase final do compilador é a geração do código objeto, consistindo normalmente de código de máquina relocável ou código de montagem. As localizações de memória são selecionadas para cada uma das variáveis usadas no programa. Então, as instruções intermediárias são, cada uma, traduzidas numa seqüência de instruções de máquina que realizam a mesma tarefa. Um aspecto crucial é a atribuição das variáveis aos registradores de memória. (Aho, Sethi e Ullman,1995)

Por exemplo:
```
mov ax,0002
mov bx,0004
add ax,bx
```
Move o valor 0002 para o registrador ax, move o valor 0004 para o registrador bx, adiciona o conteúdo dos registradores ax e bx, guardando o resultado em ax.

3.4.7 - Tabela de Símbolos

Aho, Sethi e Ullman (1995) definem que uma tabela de símbolos é usada para controlar as informações de escopo e das amarrações a respeito dos nomes. A tabela de símbolos é pesquisada a cada vez que um identificador é encontrado no programa-fonte. As mudanças ocorrem na tabela se um novo identificador ou uma nova informação a respeito de um identificador for descoberto.

Uma função do compilador é registrar os identificadores usados no programa fonte e coletar as informações sobre os seus diversos atributos. Esses atributos podem providenciar informações sobre a memória reservada para o identificador, seu tipo, escopo (onde é válido no programa) e, no caso de nomes de procedimentos, coisas tais como o número e os tipos de seus argumentos e o tipo retornado, se algum.

A tabela de símbolos é uma estrutura de dados contendo um registro para cada identificador utilizado no programa fonte além dos atributos. A estrutura de dados permite encontrar rapidamente cada registro e, igualmente, armazenar ou recuperar dados do mesmo.

A tabela de símbolos pode interagir em todas as fases num processo de compilação. Por exemplo, ao realizar a análise semântica e a geração de código intermediário, precisa-se conhecer de que tipos os identificadores são, a fim de verificar se o programa fonte os usa de forma válida.

3.4.8 - Detecção e Tratamento de Erros

Cada fase pode encontrar erros. Entretanto, após encontrá-los, precisa lidar de alguma forma com os mesmos, de tal forma que a compilação possa ou não continuar.

As fases da análise sintática e semântica tratam usualmente de uma ampla fatia de erros detectáveis pelo compilador. A fase de análise léxica pode detectá-los quando os caracteres remanescentes na entrada não formem

qualquer *token* da linguagem. Os erros, onde o fluxo de *tokens* viole as regras estruturais (sintaxe) da linguagem, são determinados pela fase de análise sintática. Durante a análise semântica, o compilador tenta detectar as construções que possuam a estrutura sintática correta, sem nenhuma preocupação

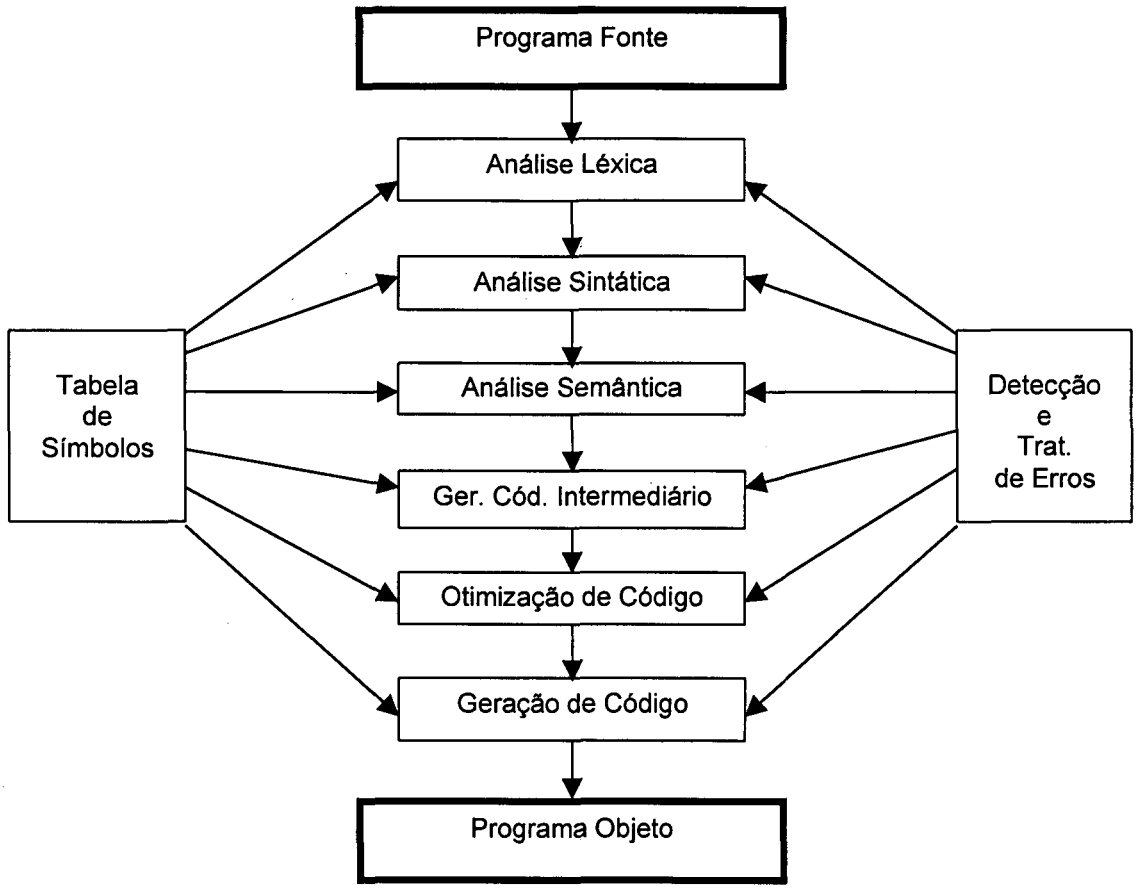


Figura 3.4 : Tabela de Símbolos e Detecção e Tratamento de Erros na construção de um Compilador.
Fonte: Aho, Sethi e Ullman, 1995, p.5.

3.5 - O Planejamento da Construção de um Compilador

Na década de 70, começaram a emergir princípios básicos eficientes para projetos de linguagens de programação. Estes princípios são fundamentais pois refletem as características ideais para desenvolvimento de uma linguagem de programação ideal.

O projeto de uma nova linguagem requer uma abordagem quanto aos aspectos que devam ser considerados no planejamento antecedendo qualquer

consideração quanto ao detalhe do projeto. Este processo tende a evitar futuros problemas. As principais características que devem ser analisadas para uma boa linguagem são:

- ambiente de programação;
- correção;
- confiabilidade;
- clareza sintática;
- clareza e concisão semântica;
- eficiência;
- portabilidade;
- simplicidade;
- ortogonalidade;
- suporte à manutenção.

O ambiente de programação, onde os programas são criados e testados, tem tido pouca influência no projeto de linguagem de programação. Entretanto, é comumente aceito que a produção de programas confiáveis e eficientes é muito mais fácil de ser conseguida, quando o trabalho é realizado com ambientes de programação apropriados.

Segundo Silva e Assis (1988), um ambiente de programação consiste basicamente em um conjunto de ferramentas de suporte e uma linguagem de comandos para invocar essas ferramentas. As ferramentas de suporte, por sua vez, são programas que podem ser utilizados pelos programadores, durante os vários estágios de criação de um programa, para auxiliá-los em sua tarefa.

A estrutura técnica de uma linguagem de programação é um dos aspectos que determinam a sua utilidade. A presença de um ambiente de programação com muitos recursos e facilidades pode tornar o trabalho atrativo, mesmo com uma linguagem tecnicamente fraca, enquanto uma linguagem tecnicamente poderosa, implementada num ambiente inadequado, pode ter seu uso inviabilizado.

Um programa é dito correto se atende plenamente suas especificações. Em contraste, um programa é confiável se for altamente provável que o serviço prestado por esse programa satisfaça seus usuários. Silva e Assis (1988) definiram que um programa pode ser confiável, mesmo que não esteja completamente enquadrado na definição de programa correto. Isto pode ocorrer quando o erro não impedir o funcionamento do sistema, podendo ser facilmente identificado e corrigido pelo usuário. Por exemplo, erros de escrita na apresentação de mensagens.

Por outro lado, programas cuja correção foi rigorosamente comprovada podem não ser confiáveis. A principal razão para esse fato reside em especificações incorretas ou que definem de forma incompleta os requisitos dos usuários (principalmente seu comportamento com relação a exceções).

Silva e Assis (1988), constataram que o suporte à confiabilidade está intimamente ligado à detecção e ao tratamento de exceções que ocorrem durante a execução de um programa. Um programa só é confiável se seu comportamento for previsível mesmo em presença de situações anômalas. Condições excepcionais como: quando houver uma divisão por zero; quando um módulo alocador de memória detectar falta de memória; quando o acesso a um *array* for feito fora de seus limites; entre outros. As limitações das linguagens no tratamento de exceções obrigam a introdução de mecanismos de testes que obscurecem a lógica central do programa. A solução mais adequada é a provisão de mecanismos de tratamento de exceção na própria linguagem de programação.

Tão importante quanto a correção sintática de uma linguagem é a capacidade da mesma em permitir a criação de programas que reflitam claramente a estrutura lógica de algoritmos. Na programação estruturada, programas são projetados hierarquicamente de cima para baixo usando um conjunto restrito de estruturas de controle (seqüenciamento, seleção e repetição). A programação estruturada, quando propriamente utilizada, produz algoritmos fáceis de serem entendidos e modificados. Uma importante característica das linguagens é a

sua capacidade de refletir, em suas construções, as estruturas de controle. Essa característica mantém a legibilidade e manutenibilidade do projeto.

A sintaxe de uma linguagem é um fator que influencia bastante a facilidade com que um programa pode ser escrito, entendido e modificado. A legibilidade de um programa é uma questão central no processo de justificativa de sua correção e em sua manutenção.

Uma linguagem deve prover um conjunto simples, claro e unificado de conceitos que o programador usa como primitivas para descrição dos algoritmos. Com essa finalidade, é desejável que as linguagens tenham o menor número possível de conceitos, associados a regras bem simples e regulares de combinações desses conceitos. Essa clareza e concisão semântica são fatores importantes para que programadores aprendam facilmente a linguagem e a usem corretamente.

A questão de eficiência tem evoluído ao longo do tempo, segundo Silva e Assis (1988). Hoje, quando se fala em eficiência de um programa, esta não pode ser medida simplesmente pelo espaço ocupado pelo código gerado ou pela sua velocidade de execução. O esforço necessário para produzir e manter um programa correto e confiável é um componente importante em qualquer medida de eficiência. E complementam:

> "A linguagem deve permitir que programas possam ser submetidos a transformações automáticas que melhorem sua eficiência. Muita pesquisa tem sido feita sobre otimização de programas e um resultado importante que tem sido demonstrado é que todas as características existentes nas linguagens que aumentam a legibilidade e a correção de programas aumentam também a sua capacidade para otimização." [Silva e Assis, 1988, p.35]

Um programa deve ter controle suficientes para ser utilizado independente da plataforma utilizada, isto é, uma linguagem é considerada independente de máquina se e somente se um programa compilado possa ser executado numa máquina X da mesma forma que numa máquina Y.

A linguagem de programação Pascal serve como exemplo para estudiosos que desejam desenvolver linguagens adequadas.

Simplicidade, ressalta Tremblay e Sorenson (1985), é a maior questão no desenvolvimento de qualquer linguagem. É evidente que a maioria dos usuários preferem uma linguagem simples. Na realidade, uma linguagem que não é simples tende a falhar em outros aspectos.

A ortogonalidade é uma característica que determina a orientação e a forma adequada para trabalhar com tipos variados de componentes, além da determinação de regras com o objetivo de proporcionar consistência no uso adequado de comandos. (Appleby, 1991)

O aspecto sobre manutenção de programas é abordado por Silva e Assis (1988) pois, segundo eles, implica a introdução de modificações e extensões por diferentes programadores em tempos distintos. Não considerando a documentação, que é fundamental em qualquer manutenção, a legibilidade dos programas é o fator mais importante no auxílio a manutenção. A capacidade das linguagens em prover abstração de dados, em suportar o método de programação estruturada e em subdividir o programa em módulos, contribui fortemente para a legibilidade dos programas.

CAPÍTULO IV

4 - PROPOSTA DE ESTRUTURA PARA UM AMBIENTE DE APRENDIZAGEM

4.1 - Introdução

Neste capítulo é descrita a proposta de estrutura para um ambiente de aprendizagem de lógica de programação.

Apesar das dificuldades encontradas pelos alunos no processo tradicional, o processo ensino-aprendizagem destas disciplinas pouco tem mudado e carece de ferramentas tecnológicas adequadas para esta atividade.

O processo tradicional de ensino decorre, posteriormente a explanação do conteúdo, das seguintes etapas:

- resolução de um problema-exemplo para apreciação de novos conceitos e/ou particularidades do problema;
- elaboração de um problema para resolução;
- acompanhamento do desenvolvimento, quando solicitado, dos que buscam ajuda ou por observação;
- apresentação de um possível resultado após um determinado tempo;
- verificação lógica do problema abstraído através do "teste de mesa";
- questionamentos quanto a alternativas de resolução e erros ocasionados no desenvolvimento.

A busca contínua do aprendizado, apesar de ser exaustivamente trabalhada, leva, consequentemente, ao surgimento dos seguintes problemas:

- o tempo dispensado para resolução do problema acaba levando os que tem facilidade a esperar a confirmação de sua solução e aos que tem dificuldades, um truncamento de seu desenvolvimento;
- dificuldades para determinar o grau de dificuldade/facilidade que cada aluno possui para resolução de problemas;
- a busca de soluções alternativas é pouco (ou quase nada) estimulada;
- a falta de atividades de recuperação no momento em que são exigidas;
- dependência da exposição da solução do problema pelo professor para validação de sua resposta.

Através dos problemas mais comuns ocorridos continuamente em sala de aula, a estrutura proposta para o ambiente de aprendizagem visa solucionar as deficiências encontradas proporcionado através de um instrumento para professores e alunos na tentativa de estimular, facilitar e aperfeiçoar estas atividades.

Papert (1985) cita que muitas vezes que a diferença evolutiva pode ser atribuída à escassez ou à ausência de materiais, a partir dos quais podem ser construídas estruturas intelectuais aparentemente "mais avançadas". A hipótese de Papert é que o computador, quando utilizado de maneira adequada, pode "concretizar" e personalizar o formal.

É justamente pela abordagem abstrata no ensino da lógica de programação que justifica-se a pesquisa e o desenvolvimento de ferramentas e/ou aplicações tecnológicas para o aprendizado ideal. Os aspectos cognitivos relacionados a captação de conteúdos comprovam a eficácia de processos que relacionam a ferramenta computador em benefício da produtividade intelectual.

Os processos cognitivos dizem respeito aos processos psicológicos envolvidos no conhecer, compreender, perceber, aprender, nas formas de pensar e nos tipos de pensamento.

Com este referencial as atividades serão apresentadas em diferentes níveis de desempenho, serão desafiadoras, pois devem estimular a procura, a busca constante e a elaboração de respostas múltiplas.

As estratégias do professor estarão centradas principalmente na iniciativa do aluno, valorizando o conhecimento que ele já traz e avançando com ele na descoberta de novas formas de trabalho.

4.2 - Recursos Necessários

Os personagens envolvidos no ambiente de aprendizagem são os mesmos de uma aula presencial: professor e aluno. Cada qual com suas características particulares e com objetivos próprios complementando-se para realização do processo ensino-aprendizagem. Através desse ambiente, o sistema deve oferecer os seguintes recursos:

Para os alunos:

- Suporte à construção individual do conhecimento;
- Facilidade para desenvolvimento do problema numa linguagem clara, própria (ou mais próxima) de sua comunicação;
- Orientação adequada e individualizada para resolução de erros;
- Acesso ao sistema de ajuda para revisão ou aprendizado de conteúdos envolvidos na disciplina;

Para os professores:

- Facilidade para elaboração e disponibilização de situações problemas;
- Facilidade para acompanhar e diagnosticar a evolução da aprendizagem;
- Avaliação das respostas dos problemas;
- Promover desafios/facilidades para cada aluno nas mensagens para resolução de erros;

Estes recursos devem estar estabelecidos mediante uma estrutura física caracterizada pelo acesso de usuários á partir de qualquer máquina ligada à rede (desde que esteja instalado o ambiente), permitindo sua plena utilização.

4.3 - Arquitetura Geral

O arquitetura geral do sistema proposto é composta por unidades fundamentais denominadas cliente e servidor. Este tipo de arquitetura é conhecida justamente como arquitetura cliente-servidor e largamente utilizada em aplicações corporativas. Uma das características básicas deste tipo de arquitetura é a diminuição do tráfego de informações pela rede.

O servidor como parte central do modelo pode associar diversas instâncias da unidade cliente. Possui a característica de assumir responsabilidades dentro do sistema tais como:

- realizar o processamento necessário para atender às solicitações dos clientes;
- servir como mediador na comunicação entre os usuários;
- armazenar todos os dados inerentes ao sistema;

A arquitetura geral do ambiente proposto é mostrado na figura abaixo.

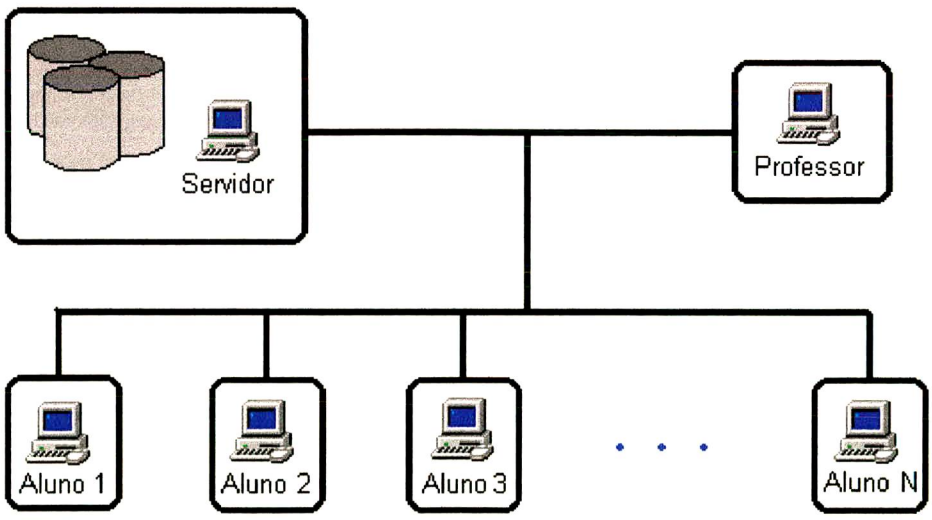


Figura 4.1: Arquitetura Geral do Sistema.

4.4 - Modelo do Ambiente de Aprendizagem

O modelo do ambiente deve estar estabelecido de forma a proporcionar a utilização do ABC-Pro em prol da atividades de ensino de lógica de programação fazendo com que a relação professor-aluno, independente da distância física, se torne mais integrada.

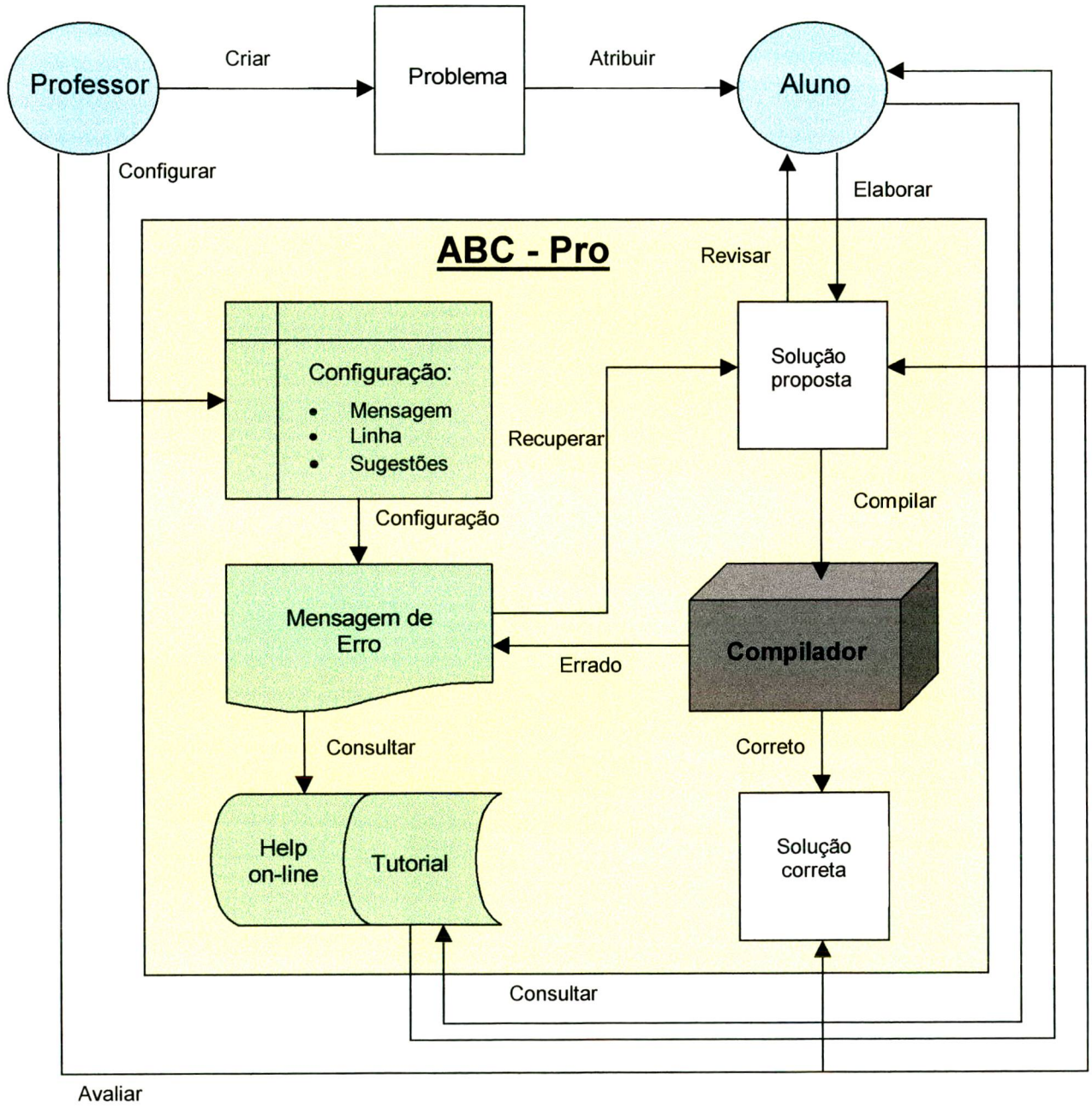


Figura 4.2: Modelo do Ambiente de Aprendizagem

A aprendizagem de lógica de programação, segundo Koslosky (1999), deve levar em consideração a bagagem de conhecimentos e conceitos que os alunos já trazem de outras disciplinas do currículo de segundo grau, especialmente os de matemática e português. De posse do problema a ser solucionado o aluno pode desenvolver seu raciocínio lógico na forma de algoritmo estruturado ou na linguagem de programação que estiver estudando no momento. No ambiente proposto, esta premissa se ajusta às necessidades básicas de aprendizagem.

4.4.1 - Relação Professor-Aluno

A relação professor-aluno parte do pressuposto básico no processo ensino-aprendizagem permitindo a transmissão do conteúdo, experiências e problemas para aplicação e construção do conhecimento pelos alunos. Até então, tradicionalmente aplicados.

Na relação professor-aluno, a mediação na aprendizagem se caracteriza através do estímulo gerado pelo professor e absorvido pelo aluno através da motivação. O estímulo gerado age sobre os alunos de modo que eles organizem e desenvolvam seu próprio conhecimento de maneira cada vez mais elaborada.

O desenvolvimento cognitivo do indivíduo é resultado combinado da exposição direta ao mundo e da experiência mediada. São as ações mútuas entre o indivíduo e o meio pela qual a cultura se transmite.

O objetivo de professor na mediação deve ser aumentar a atividade cognitiva do aluno fazendo que reflitam sobre seus processos abordando sistematicamente soluções de problemas de forma estratégica. O professor mediador deve fazer com que o aluno seja mais eficiente em aprender como aprender, generalizando os processos de solução de problemas.

4.4.2 - Relação Aluno-Ambiente

A relação aluno-ambiente oportuniza meios pelos quais o aluno desenvolverá soluções de situações-problemas encaminhadas pelo professor.

O desenvolvimento da inteligência é determinado pelas ações mútuas entre professor-aluno-ambiente. O aluno responde aos estímulos externos do professor agindo sobre eles para construir e organizar o seu próprio conhecimento. Com o problema atribuído pelo professor, o aluno elabora seu programa diretamente ao ambiente e encaminha-o ao processo de análise pelo compilador. O aluno aprende melhor quando toma parte de forma direta na construção do conhecimento que adquire.

O ABC-Pro, neste contexto, atuará como supervisor no desenvolvimento destas soluções. O processo fundamental do ABC-Pro, consiste num compilador que verificará a solução proposta pelo aluno. Nesta interação estará estabelecido um meio pelo qual o ambiente se comunicará com seu usuário (no caso o aluno) dando como possivelmente concluída a resolução do problema (se o problema estiver correto), ou indicando falhas nas instruções estabelecidas. Estes problemas serão apresentados para o aluno que o tratará ou requisitará a exibição de um sistema de ajuda (help on-line) que lhe permita o esclarecimento mais detalhado. O ABC-Pro permite, em qualquer situação, consultar determinado conteúdo da disciplina através do módulo tutorial. Este processo gera um ciclo que termina com uma possível solução correta.

Com estas atividades o aluno retém o conhecimento através de esquemas pois fornecem base de referência que ajudam a compreender e lembrar de situações passadas experienciadas. Os esquemas servem como estrutura do conhecimento organizado ao qual novos eventos podem ser relacionados.

4.4.2 - Relação Professor-Ambiente

O encaminhamento da solução-proposta pelo aluno deve permitir ao professor detectar quais mecanismos cognitivos devem ser acionados para superar o

impasse em que o aluno se encontra. Para o professor, o ABC-Pro proporcionará facilidades para detectar os alunos ou "pontos" que deverão ser mais trabalhados. Este processo poderá ser feito mediante avaliação da solução proposta ou correta.

Outro contexto de mediação na utilização do ambiente, considerando uma devida avaliação, o professor poderá interagir no ambiente do aluno de forma a proporcionar ou retirar aspectos esclarecedores das mensagens de erro que lhe é proporcionado. O professor poderá, em função do aprimoramento técnico, proporcionar desafios para resoluções de erros. Esta atividade faz com que o aluno tenha que dominar as questões técnicas e lógicas definidas para a resolver um problema.

O professor, como mediador do conhecimento, poderá atribuir problemas aos alunos de forma individualizada, respeitando o avanço e a velocidade de aprendizagem de cada um, podendo diminuir ou aumentar o grau de complexidade das situações-problema trabalhadas.

CAPÍTULO V

5- IMPLEMENTAÇÃO DO ABC-PRO

5.1 - Introdução

Neste capítulo é descrita a implementação do ambiente ABC-Pro, cujo protótipo foi desenvolvido em Delphi. O Delphi é uma ferramenta que possibilita o desenvolvimento de aplicações visuais, garantindo, consequentemente que toda e qualquer aplicação tenha critérios satisfatórios de interface. Para este trabalho a interface e os requisitos para manipulação de arquivos textos são importantes para a melhor utilização e armazenamento de programas. O Delphi permite, como destaque, facilidade e rapidez no desenvolvimento da interface, facilidade para criação de aplicativos cliente-servidor e suporte para uso em intranet corporativa.

O planejamento para a construção do ambiente (sua estrutura, recursos e utilidade), bem como a construção do compilador é a circunstância básica necessária para o sucesso deste protótipo.

Os detalhes técnicos quanto ao desenvolvimento do compilador permitem esclarecimentos específicos com relação a forma de implementação e a fases de análise desenvolvida. Além disso, o tratamento de erros esta voltado para questões pedagógicas. O esclarecimento de erros e o conteúdo sobre lógica de programação estão devidamente incorporados ao sistema.

5.2 - Planejamento

5.2.1- Interface

A interface é a parte do sistema computacional com a qual o usuário entra em contato por meio do plano físico, perceptivo e cognitivo. Em outras palavras, a interface é uma fronteira comum entre o computador e o homem. O objeto principal de estudo é o homem. Mas, o objeto de interesse prático é o computador. (Andrade, 1998)

Se a proposta é oferecer um ambiente que favoreça o aprendizado da lógica de programação, obviamente este ambiente deve ter condições suficientes para que o aluno interaja naturalmente com os recursos advindos do sistema.

As aplicações visuais, além de possuírem interfaces amigáveis, possuem processos comuns. Estes processos permitem facilidades na utilização destas aplicações pois o procedimento empregado em uma aplicação podem ser a mesma em outra. Por exemplo: o processo para abrir um arquivo, quase sempre será feito da mesma forma, ou seja, clicando-se na opção Arquivo e depois Abrir (ou *File - Open*). Pode-se imaginar uma série de programas aplicativos que possuem este processo entre outros recursos.

Estabeleceu-se uma padronização quanto ao uso do ambiente de programação. Os componentes distribuídos adequadamente que compõem a interface (como mostra a figura 5.1) são:

- menu de aplicação
- barra de ferramenta
- área de desenvolvimento
- barra de *status*

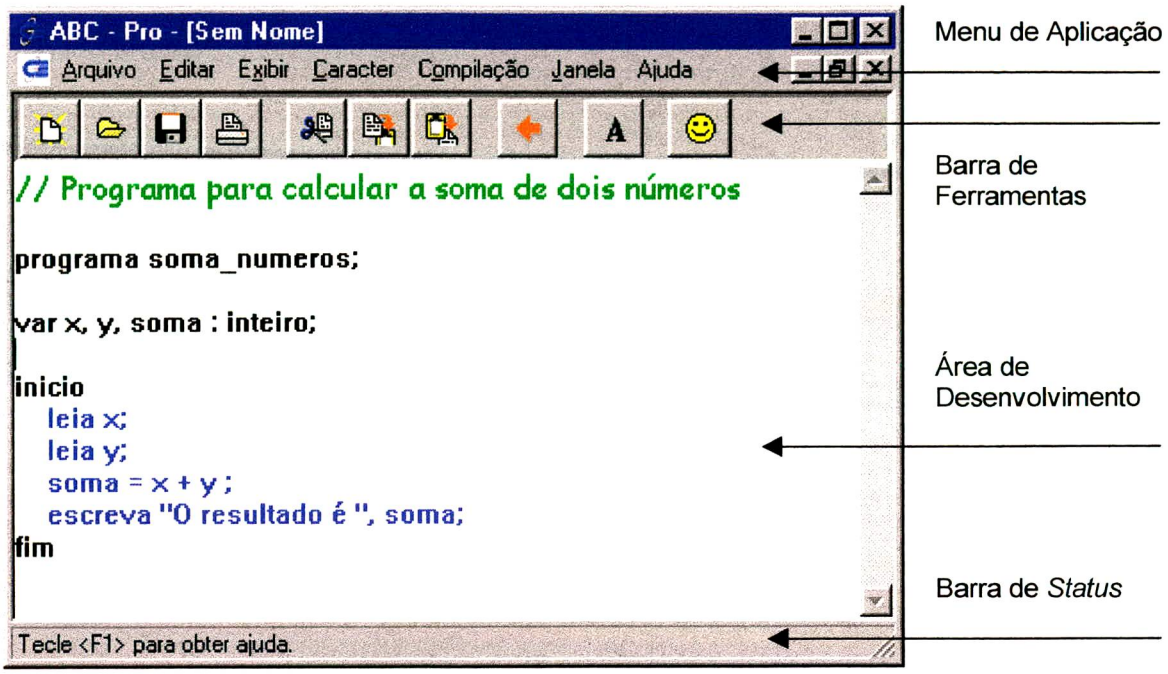


Figura 5.1 – O Ambiente de Programação.

O menu de aplicação foi definido e está implementado de forma a proporcionar manipulação semelhante aos programas aplicativos no qual se costuma trabalhar juntamente com os recursos técnicos do sistema. As opções estão distribuídas como segue:

- Arquivo: Novo, Abrir, Fechar, Salvar, Salvar Como, Imprimir, Configurar Impressora e Sair;
- Editar: Desfazer, Recortar, Copiar, Colar, Limpar, Selecionar Tudo;
- Exibir: Barra de Ferramentas, Gênio, Erros (Mensagem, Linha de Ocorrência, Sugestões), Lista de *Tokens*, Tabela de Símbolos, Tabela de Código Intermediário;
- Caracter: Esquerda, Direita, Centralizado, *Word Wrap*, Fonte;
- Compilação: Compilar, Analisador Léxico, Analisador Sintático e Semântico;
- Janela: Lado a Lado, Cascata, *arquivos*;
- Ajuda: Como Usar, Guia do Aprendizado, Sobre o ABC-Pro;

Existe, ainda, o menu *pop-up* que possibilita o uso dos procedimentos: Recortar, Copiar, Colar e Fonte.

A barra de ferramentas consiste na invocação de processos mais utilizados. São eles: Novo, Abrir, Salvar, Imprimir, Recortar, Copiar, Colar, Desfazer, Fonte e Compilar.

Inovações de recursos surgem, cada vez mais, aprimorando a interface do Usuário. Com o lançamento do Microsoft Office 95, muitos usuários ficaram empolgados com uma animação que ajuda a ensinar o modo certo de realizar as operações. Este recurso, chamado de Microsoft Agent, implementa serviços que permite que a programação de personagens animados dentro de sua própria aplicação. Com o MSAgents criou-se uma alternativa no processo de emissão de mensagens permitindo maior interatividade.

5.2.2 - Do Compilador

Apesar de estar bem definido todas as fases envolvidas no processo de compilação é importante ressaltar uma fase que antecede ao processo de construção que é o planejamento de um ambiente de linguagem de programação.

A linguagem estabelecida consiste fundamentalmente em terminologias conhecidas para a língua portuguesa. Além disso, estabeleceu-se uma gramática apropriada que tenha características de alguma linguagem usual para que na transição futura o aluno não sofra tanto com inconvenientes de uma nova linguagem.

Os detalhes técnicos da implementação e planejamento estão descritos nas fases de construção do compilador.

5.3 - Construção do Compilador

Diante de um planejamento devidamente analisado, está esclarecido na seqüência todos os critérios e fases estabelecidos para a implementação do compilador.

5.3.1 - Forma de Implementação

A quantidade de vezes que o programa fonte é analisado está definida com o objetivo de promover uma ordem no nível de supostos erros que provavelmente venham a ocorrer.

O compilador realiza o processamento em 2 passos. O primeiro passo consiste na análise léxica com o objetivo de identificar todos os *tokens* utilizados no programa. Este processo define inicialmente que todas as "palavras do vocabulário" sejam reconhecíveis. É como se verificasse primeiro a existência de "erros ortográficos".

O segundo passo realiza o restante das fases. A lista de *tokens* (anexo 1) serve como entrada para fase de análise sintática. Uma conseqüente atividade da análise sintática é de promover, também, a análise semântica na qual estes procedimentos estão embutidos na gramática estabelecida.

Como estão estabelecidas as ações semânticas na gramática do compilador (ver anexo 2, os itens com '#'), pode-se desenvolver ações para Geração de Código Intermediário. A partir daí, o processo de síntese do compilador poderá ser executado permitindo o desenvolvimento futuro do programa executável.

5.3.2 - Análise Léxica

Como definido no capítulo 3, o processo de análise léxica tem por objetivo a verificação de todos os itens léxicos ou *tokens* que compõem o programa fonte avaliando se cada item é válido. Entretanto, nesta fase definiu-se as regras nas quais os itens léxicos devem respeitar, ou seja, quais os tipos de "palavras" que nossa linguagem aceita.

Inicialmente, destaca-se as classes de *tokens* que são utilizadas no ambiente de acordo com sua funcionalidade, a saber:

- Identificadores;
- Palavras Reservadas;

- Constantes numéricas;
- Constantes literais;
- Símbolos Especiais.

Os **identificadores** são utilizados no programa para denominação de variáveis, constantes ou nome do programa. Os identificadores (figura 5.2), são reconhecidos a partir de uma letra até encontrar um caracter não aceito para sua identificação, isto é, um caracter diferente de letra, dígito ou *underline* (_). Exemplo de identificadores: nome, nota_1.

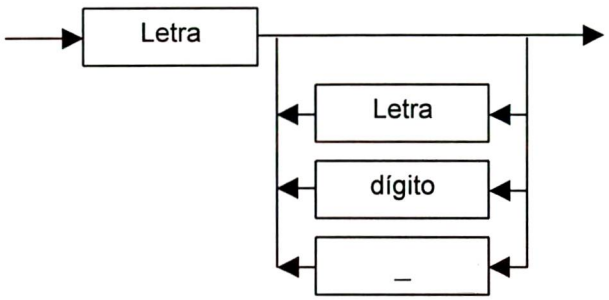


Figura 5.2 – Diagrama de reconhecimento de identificadores.

Por letra entende-se qualquer caracter entre 'A' à 'Z' e 'a' à 'z' e por dígito qualquer caracter entre '0' e '9'. Não há limite quanto ao tamanho do identificador justamente para que estes mesmos identificadores sejam utilizados da forma mais clara possível. A prática, quanto a utilização de identificadores, mostra que a tendência é tê-los com tamanho reduzido.

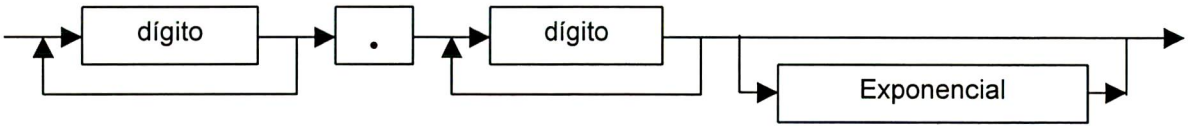
Na categoria dos identificadores deve-se avaliar cada ocorrência verificando a possibilidade deste identificador ser uma palavra reservada, ou seja, uma palavra pertencente ao vocabulário da gramática como por exemplo: inteiro, real, leia, escreva.

Se o identificador pertence ao conjunto de palavras reservadas atribui-se um *token* específico da referida palavra, caso contrário, atribui-se um *token* representando um identificador. No anexo 1, encontra-se a relação de todos os itens léxicos envolvidos no projeto, bem como, os *tokens* gerados e o seu significado.

Constantes numéricas são informações que representam valores numéricos. As constantes numéricas são reconhecidas à partir de um dígito. Há dois *tokens* possíveis para estas constantes: para constantes numéricas inteiras e reais. Por exemplo: 35, 103.57, 8E-5. É possível utilizar a notação matemática para números exponenciais (8E-5 eqüivale à 0.00008).

A figura (5.3) mostra os diagramas no processo de reconhecimento destes *tokens*.

- Constante Numérica Real



- Constante Numérica Inteira

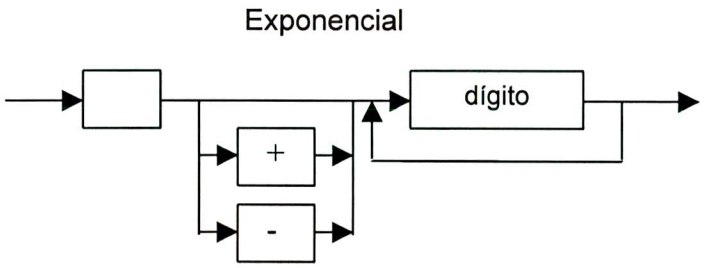


Figura 5.3: Diagramas de Constantes Numéricas Real e Inteira.

Os **literais** ou constantes literais são qualquer seqüência de caracter que esteja entre aspas duplas ("). Por exemplo: "UFSC". A figura 5.4 mostra o diagrama para o reconhecimento de literais.

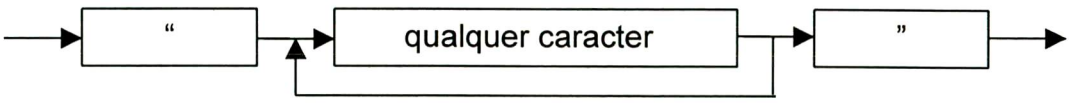


Figura 5.4: Diagrama de Literal.

Os **símbolos especiais** são cuidadosamente tratados para que não haja uma dissonância com relação a sua utilização. Na maioria das linguagens adota-se

os mesmos tipos de símbolos especiais. Por exemplo: + (adição), * (multiplicação), / (divisão), > (maior), <> (diferente), etc.

Os símbolos especiais são utilizados para operadores (aritméticos, relacionais e concatenação de literais), comando de atribuição, pontuação de fim de comando e declarações de constantes ou variáveis, prioridades em expressões, etc.

Os símbolos especiais estão divididos em símbolos simples e símbolos duplos como consta em anexo 1.

Os espaços em branco e caracteres de controle são ignorados pelo compilador assim como os comentários, ou seja, a análise léxica não gera *tokens* para estas representações. O objetivo do comentário é esclarecer o conteúdo do programa para facilitar seu entendimento. Para o analisador léxico, os comentários iniciam com duas barras (//) e se estende até o fim de sua linha. Isto implica que pode-se ter em uma linha um comando seguido de comentário. Por exemplo: media = (n1 + n2) / 2 ; //cálculo da média.

Qualquer existência de caracteres não relatados acima - considerando que não faça parte de um comentário ou literal - constituirá num erro léxico, bem como, um literal sem as aspas finais ou constantes numéricas que não respeitam sua estrutura.

Exemplo de erros léxicos:

.45	- constante numérica incompleta
&	- caracter inválido
"este literal não tem fim	- literal sem fim
5E+	- constante numérica incompleta

Desta forma, o processo de compilação identifica inicialmente os *tokens* gerados, bem como a existência de erros léxicos. Isto implica inicialmente na correção de erros ortográficos, ou seja, "palavras" que não fazem parte do vocabulário. São as regras iniciais estabelecidas para a linguagem ABC-Pro.

As técnicas estabelecidas para o reconhecimento de *tokens* são: os autômatos finitos e expressões regulares.

A expressão regular é uma notação que permite definir precisamente o conjunto daquela natureza. Com esta notação, definiu-se os identificadores como: letra (letra | dígito | '_') * e as constantes numéricas como: dígito $^{+}$ (. dígito $^{+}$) $^{?}$ ('E' (+ | -) $^{?}$ dígito $^{+}$) $^{?}$. O sinal de + significa o acontecimento do conteúdo uma ou mais vezes; a ? uma ou nenhuma vez e a barra vertical indicando uma disjunção.

O AF utiliza como representação gráfica o diagrama de transição (figura 5.5) que é um grafo direcionado e rotulado, onde os vértices representam os estados e fisicamente são representados por círculos (sendo que os estados finais são representados por círculos duplos), e as arestas representam as transições.

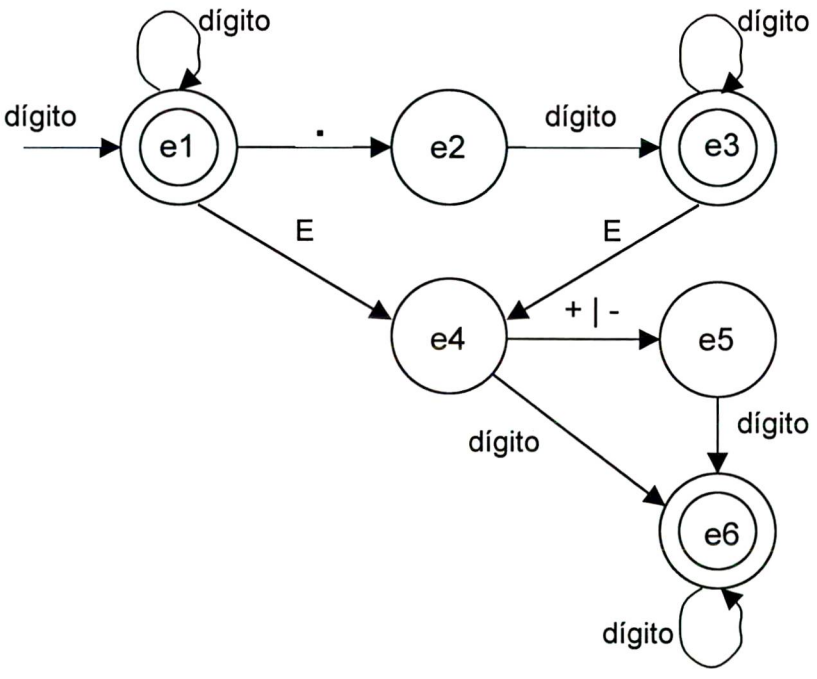


Figura 5.5: Autômato Finito de Constantes Numéricas.

Outra representação utilizada é a tabela de transições (tabela 5.1) que representam de forma tabular um AF. Nesta tabela, as linhas representam os estados (o inicial é indicado por uma seta e os finais por *asterísticos*), as

colunas representam os símbolos de entrada e o conteúdo de uma posição é a transição.

	dígito	.	E	+	-
→ * e1	e1	e2	e4		
e2	e3				
* e3	e3		e4		
e4	e6			e5	e5
e5	e6				
* e6	e6				

Tabela 5.1: Tabela de Transição para Constantes Numéricas.

A ocorrência de um estado não previsto, considerando que o estado anterior não é um estado final, implica, consequentemente, em erro.

5.3.3 - Análise Sintática

A fase da análise sintática, como visto no capítulo 3, consiste na verificação da ordem na qual os *tokens* estão dispostos respeitando desta forma as regras regidas pela gramática. O anexo 2 mostra a gramática estabelecida para o ABC-Pro.

A especificação sintática da linguagem de programação está baseado no planejamento do compilador determinados pelo objetivo, filosofia e potencialidades. A simplicidade da linguagem do ABC-Pro é uma das maiores evidências. Antes de qualquer coisa, a estrutura de desenvolvimento e comandos são simples justamente para facilitar o desenvolvimento de programas. A complexidade de comandos existentes é a menor possível, justamente para facilitar o entendimento pelo aluno.

A clareza sintática do ABC-Pro influência bastante na facilidade com que um programa pode ser escrito, entendido e modificado. Permite o desenvolvimento de programas numa forma algorítmica, ou seja, praticamente, a forma desencadeada na resolução do problema. É a capacidade de refletir facilmente suas construções. Os programas são legíveis, fácil de entender e desenvolver.

Na especificação preliminar ficou determinado de maneira informal as construções que seriam necessárias na linguagem como a estrutura do programa, os tipos de declarações e os comandos.

O tipo de estrutura para desenvolvimento da gramática adotado foi a Gramática Livre de Contexto (GLC) por caracterizar-se pela eficiência e objetividade de implementação.

Segundo Furtado (1987), dentre os quatro tipos de gramáticas, as GLC são as mais importantes na área de compiladores e linguagens de programação pelo fato de especificarem eficientemente as construções sintáticas usuais.

Aho, Sethi e Ullman (1995) identificam que para especificar a sintaxe de uma linguagem, uma notação amplamente aceita é a chamada Gramática Livre de Contexto ou BNF. Uma GLC possui quatro componentes:

- Um conjunto de *tokens*, conhecidos como símbolos terminais;
- Um conjunto de não terminais;
- Um conjunto de produções;
- Uma designação a um dos não-terminais como o símbolo de partida.

O anexo 2 mostra estes conjuntos definidos para o ABC-Pro. Perceba que tem-se o não-terminal <programa> como símbolo de partida que gera a produção: programa *id* ';' <bloco>. Esta produção é o ponto de partida para o reconhecimento de qualquer programa desenvolvido no ambiente. Inicialmente, com a palavra reservada PROGRAMA; um identificador qualquer referenciado como *id*; o símbolo especial simples ponto-e-vírgula (;); e na continuação a invocação de um outro não-terminal (<bloco>) que gera uma produção com mais dois não-terminais e assim sucessivamente.

Como apoio ao desenvolvimento da análise sintática, utilizou-se a ferramenta GAS (Gerador de Analisador Sintático) para a implementação do programa responsável pela verificação sintática do programa.

O GAS foi desenvolvido como projeto de conclusão de curso de ciências da computação na UFSC. Dentre os principais aspectos destaca-se: as diversas técnicas de análises sintáticas; gerador código para as linguagens pascal, c e módula 2; possui simulador; gerador de documentação.

5.3.4 - Análise Semântica

A análise semântica verifica a lógica do programa avaliado se existe significado nas composições sintáticas.

É importante que os alunos aprendam facilmente a linguagem e a usem corretamente. Além de prover um conjunto claro, simples e reduzido de conceitos e comandos para desenvolvimento de programas, é importante a associação de regras simples e regulares para combinação desses conceitos.

As rotinas que verificam a semântica do programa estão incorporadas na análise sintática, isto é, nas produções da gramática existem as ações semânticas (ver anexo 2) definidas com '#' e um número correspondente à ação a ser realizada.

As funções do analisador semântico (ver anexo 3) consiste nos seguintes procedimentos:

- Verificação no uso de identificadores:
  - O identificador utilizado como denominador do programa não pode ser utilizado posteriormente.
  - Nas declarações, um identificador não pode ser declarado duplamente;
  - Não pode ser utilizado no corpo do programa se não estiver devidamente declarado;
  - Nas declarações de constantes, deve ser atribuído um número real ou inteiro, um literal ou os valores lógicos verdadeiro ou falso
  - Os tipos utilizados para declaração de variáveis devem ser: inteiro, real, caracter ou *booleano*;
  - Os identificadores declarados como constante não podem ser utilizados em comandos nos quais lhe altera o conteúdo.

- Nas expressões e atribuições devem ser utilizados elementos do mesmo tipo, isto é, deve haver compatibilidade entre estes elementos. A única exceção, naturalmente, é a utilização em uma expressão de elementos do tipo inteiro e real;
- No comando de atribuição uma variável real aceita uma informação do tipo inteiro;
- Em expressões utilizadas em estruturas de controle (se e enquanto), o resultado gerado deve ser lógico (verdadeiro ou falso);
- Na estrutura de repetição para-faça os limites inicial e final devem ser do tipo inteiro.

Neste processo a implementação da tabela de símbolos torna-se imprescindível. A tabela de símbolos é utilizada inicialmente na fase de análise semântica para especificar os identificadores que são inicialmente declarados, além de sua categoria (nome do programa, constante ou variável), valor atribuídos a estes identificadores (se houver) e seu tipo de dado. Estas informações são fundamentais pois a cada reconhecimento de identificador, a tabela é pesquisada com o objetivo de validar ou não o uso deste identificador.

5.4 - O Sistema de Ajuda

5.4.1 - O Tutorial

Com o objetivo de proporcionar assistência no aprendizado criou-se um módulo no sistema de ajuda no ABC-Pro que permite ao aluno rever conceitos fundamentais do conteúdo sobre lógica de programação.

Este ambiente proporciona uma "navegação" seqüencial ou uma consulta no ponto de interesse. Está distribuído em módulos organizados de forma a possibilitar um grau progressivo no processo de aprendizagem. As figuras 5.6 e 5.7 mostram a organização deste tutorial chamado de "Guia do Aprendizado".

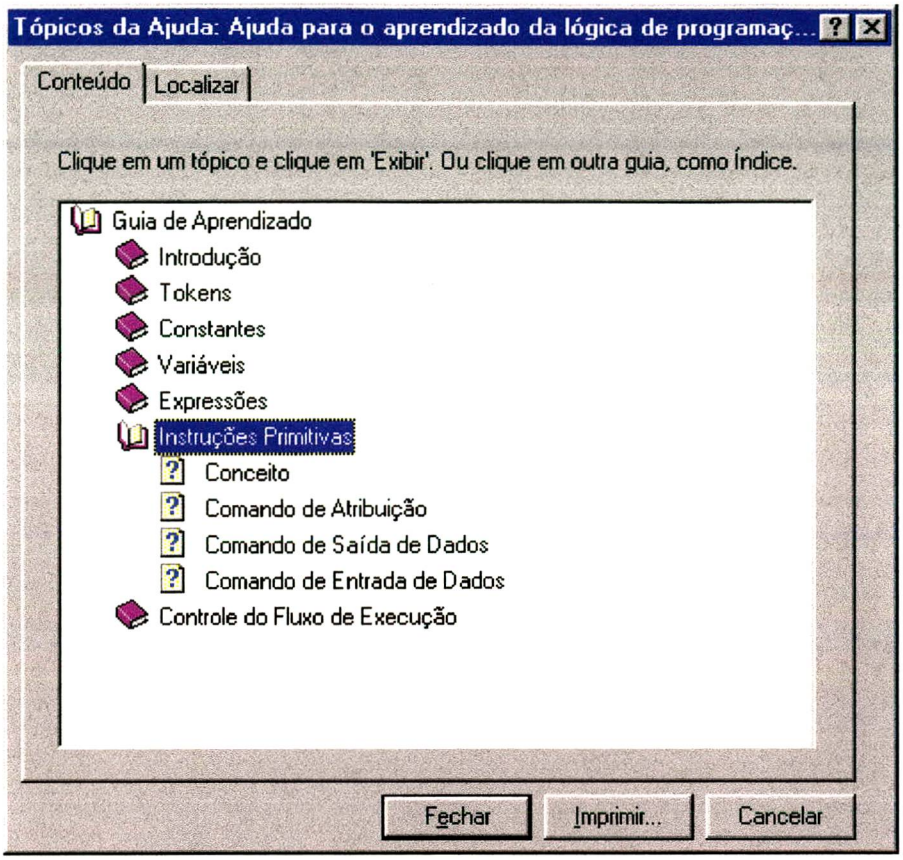


Figura 5.6: Estrutura do Guia do Aprendizado.

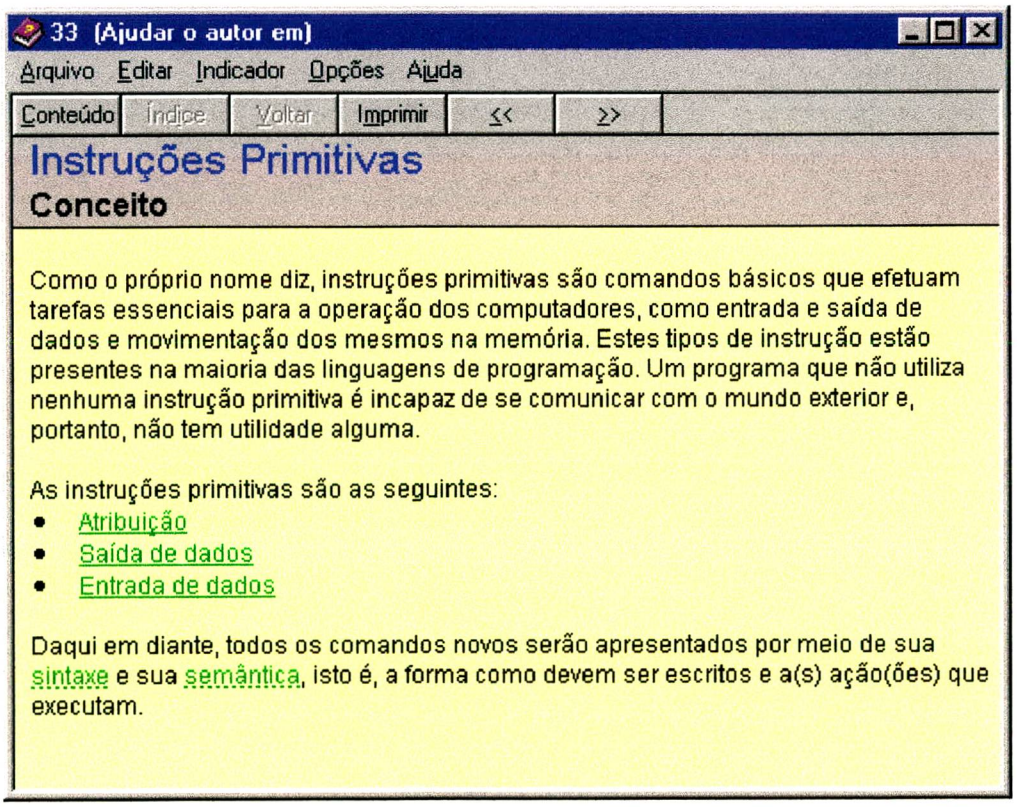


Figura 5.7: Conteúdo de um dos tópicos do Guia do Aprendizado.

O aluno possui autonomia quanto à utilização do tutorial permitindo que o conhecimento adquirido possa ser testado no ambiente de programação para validação de seu aprendizado. Isto implica fundamentalmente que o aluno busque novas informações em benefício próprio. Quanto mais for aprendido e validado, maior a motivação e segurança pois, consequentemente, a formação de esquemas nas estruturas cognitivas estarão sendo estabelecidos para posteriores desenvolvimento de programas de complexidades variadas.

5.4.2 - Help On-line

O Help On-line proporciona, mediante a não resolução de algum erro, o acesso e o esclarecimento pertinente ao erro. Esta atividade permite autonomia para detecção de possíveis falhas técnicas quanto ao desenvolvimento de programas oferecendo oportunidades para que o aluno entenda porque errou. Descobrindo por si próprio através da investigação o aluno aprende por descoberta.

O objetivo consiste em converter a dúvida pelo aprendizado. Nas concepções do ABC-Pro considera-se o erro como importante fator de aprendizagem.

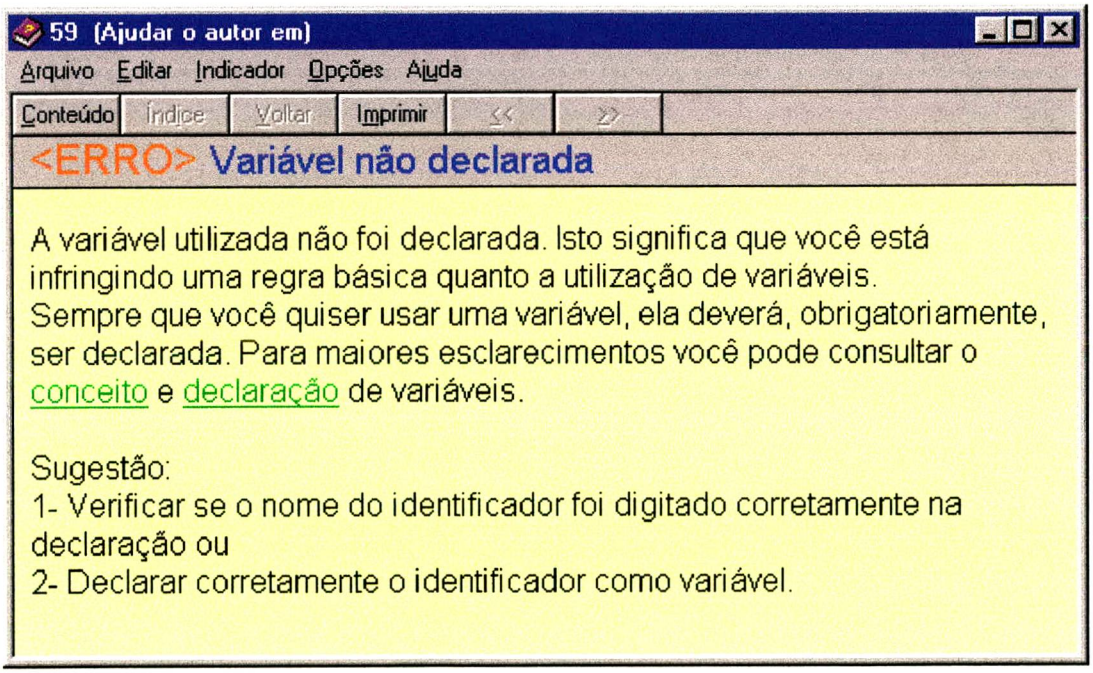


Figura 5.8: Help on-line na resolução de um erro.

5.4.3 - O Mediador X Help

Mediar na aprendizagem implica no estabelecimento de técnicas adequadas ajustadas de forma a viabilizar o melhor caminho do aluno ao conhecimento. O objetivo do mediador é promover a motivação na construção do conhecimento fazendo com que a reutilização de esquemas proporcionem acomodação através de novos esquemas. Na tentativa de promover o aprendizado através do raciocínio, mediante a experimentação e a pesquisa, existe um módulo para gerenciar o grau de assistência que o aluno poderá dispor. Através deste módulo será possível estabelecer desafios aos alunos através da omissão das possíveis mensagens de erros.

A complexidade dos problemas apresenta fatores importantes a serem considerados: pode ser estranho para alguns, absorvendo assim maior energia e atenção, precisando de um tempo adicional para solucionar o problema e dominar o conteúdo. Por outro lado, pode ser muito familiar, ou até banal, podendo desmotivar o aluno para o trabalho de resolução do problema.

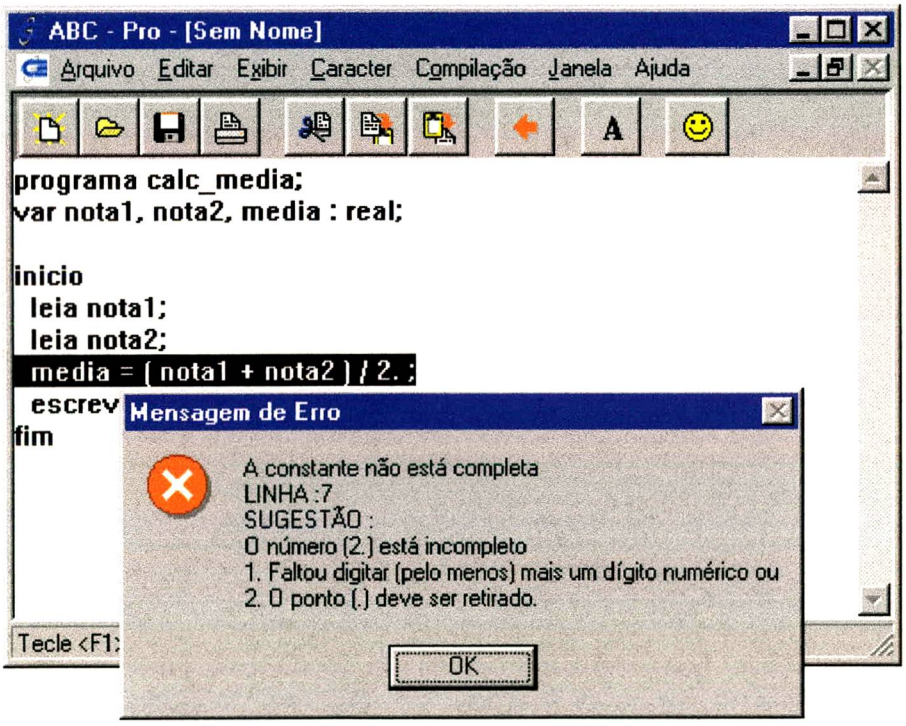


Figura 5.9: Exibição de mensagem de erro.

O objetivo deste módulo é proporcionar, quando da ocorrência de erros no programa, a apresentação de uma caixa de dialogo (ou através do agente) com informações como: a mensagem sobre o erro acontecido, a linha de programa onde ocorreu o erro (além da seleção desta linha) e sugestões onde pode-se ter possíveis soluções para o erro encontrado. As figuras 5.9 e 5.12 mostram como isto funciona utilizando caixa de diálogo e o Agente Gênio, respectivamente.

A possibilidade mais importante neste processo é o grau de assistência que o aluno pode ter. No início do aprendizado da lógica de programação, é natural que todas estas opções estejam habilitadas. Entretanto, no decorrer da evolução do conteúdo, o professor-mediador poderá estabelecer um grau maior de dificuldade para a resolução desses erros. O objetivo é promover o conhecimento através de seus próprios erros, pois à medida que aumenta a dificuldade através de menos assistências, maior deverá ser o grau de raciocínio para resolução de erros e, consequentemente, a qualidade do aprendizado será significativa. Esta configuração quanto às informações que podem ser apresentadas estão dispostas no menu Exibir - Erros como mostra a figura 5.10.

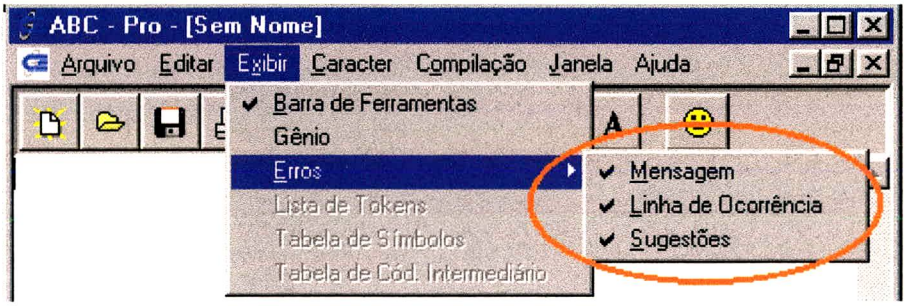


Figura 5.10: Configuração das mensagens de erros a serem exibidas.

Uma alternativa na exibição das mensagens no sistema é o uso do Microsoft Agent. No ABC-Pro, habilitou-se o agente chamado: Gênio. Este agente possui característica reativa e é utilizado apenas como alternativa na interface permitindo uma interatividade mais agradável nas situações de erro. A habilitação do agente é definida pelo usuário através do opção Exibir - Gênio.

Figura 5.11: Apresentação do agente Gênio.

O agente interage (se estiver habilitado) com o aluno sempre que houver um processo de compilação que implica, consequentemente, em uma emissão de mensagem. Em situações de erro o agente entristece e fornece-lhe a mensagem (figura 5.12) incentivando-o na busca da solução. Enquanto habilitado, o agente fica visível executando atividades aleatórias sem interferir no desenvolvimento do programa.

Figura 5.12: Exibição de mensagem de erro pelo agente.

A utilização do agente pode acompanhar processos alternativos como emissão de som. Este recurso vem de encontro com as pesquisas de Gardner, permitindo alternativas para beneficiar o processo na formação das estruturas cognitivas. O aluno escolhe a forma mais adequada para interagir com o sistema.

Novas tecnologias de *hardware* e *software* (processadores mais possantes, placas de vídeo com muita memória e alta resolução, placas de som estéreo, telas sensitivas ao toque, sintetizadores de voz, capacidade de reconhecimento de voz e escrita, teoria de agentes, realidade virtual, etc.) podem proporcionar interfaces mais próximas para comunicação e entendimento do conteúdo.

A utilização do ABC-Pro, além de permitir a implementação de programas numa linguagem algorítmica com suporte a tratamento de erros e sistemas de ajuda, contempla (de forma transparente para o usuário) técnicas adequadas para facilitar a utilização e a exploração do ambiente.

Além da preocupação técnica que consiste no desenvolvimento do compilador para algoritmos e sistemas de ajuda, buscou-se alternativas para viabilização de uma interface que pudesse oferecer ao aluno, melhores (e mais agradáveis) condições para utilização do ABC-Pro. Com isso, minimiza-se os problemas de adaptação ao ambiente e o tempo para gerar os primeiros resultados, maximizando, consequentemente, o aprendizado

CAPÍTULO VI

6 - APLICAÇÃO NO CEFET/SC – CENTRO DE FORMAÇÃO TECNOLÓGICA

6.1 – História do CEFET/SC

Em 1° de setembro de 1910 instala-se em Florianópolis, num prédio situado à rua Dr. Victor Konder, a então Escola de Aprendizes de Artífices de Santa Catarina, criada através do decreto no 7566 de 23/10/1909.

Com matrícula inicial de 100 alunos, começa sua atuação junto a comunidade, oferecendo habilitação nas áreas de ferraria, serralheria, mecânica, carpintaria, encadernação e tipografia.

Dez anos após é transferida para um prédio situado à rua Almirante Alvim, onde permanece até 1962, quando passou a funcionar no prédio atual, à Av. Mauro Ramos, 950.

Ao longo do tempo passou por sucessivas e importantes mudanças estruturais, fazendo com que sua denominação também se alterasse:

- Liceu Industrial de Florianópolis em 1937;
- Escola Industrial de Florianópolis em 1942;
- Escola Industrial Federal de Santa Catarina em 1965;
- Escola Técnica Federal de Santa Catarina em 1968.

Finalmente, através da lei no 8948 de 1994, a ETF/SC é transformada em Centro Federal de Educação Tecnológica – CEFET/SC, possuindo Unidades de Ensino sediadas em Florianópolis, São José, Jaraguá do Sul e Joinville.

Ao longo de sua existência, esta instituição teve sempre como objetivo garantir aos seus alunos uma formação profissional em sintonia com as transformações e avanços tecnológicos associados às exigências do mercado de trabalho.

Através de seu programa de qualidade total, incluindo o planejamento estratégico, a missão da Escola está definida como: Gerar e difundir conhecimento tecnológico e formar indivíduos capacitados para o exercício da cidadania e da profissão.

6.2 - Curso Técnico de Informática

O Curso Técnico de Informática foi criado em agosto de 1995 e, primeiramente, passou a ser ministrado na cidade de Xanxerê. Isto só foi possível em virtude de um convênio FETESC/UNOESC, onde a primeira era responsável pelas atividades didáticos/pedagógicas e a segunda pela estrutura física. À partir de agosto de 1996, passou também a ser ministrado na Unidade de Ensino de Florianópolis, como curso regular do sistema CEFET/SC.

O objetivo do Curso Técnico de Informática é proporcionar formação profissional na área de informática para atuação na administração pública ou privada, desenvolver e incentivar o espírito empreendedor, fazendo com que o profissional egresso tenha como alternativa a criação do seu próprio negócio, adaptando deste modo metodologias de ensino às exigências de um mercado cada dia mais competitivo.

A coordenação do curso no que se refere as atividades didático/pedagógicas estão sob a supervisão do Núcleo de Informática e Sistemas (NIS), subordinado a Gerência de Formação Geral e Serviços na estrutura organizacional da Unidade de Ensino de Florianópolis.

O NIS conta atualmente com nove professores que ministram as disciplinas do curso e também disciplinas de informática para os demais cursos do CEFET/SC – Florianópolis.

A carga horária média é de 20 horas/aula, sendo também responsáveis por outras atividades correlatas tais como pesquisa, extensão, supervisão de estágio, análise e avaliação de relatórios.

A relação completa das disciplinas ministradas pode ser conhecida através de tabela abaixo.

MATÉRIAS	DISCIPLINAS	CARGA SEMANAL			SUBTOTAIS	
		1º SEM	2º SEM	3º SEM	CS	h/a
LÍNGUA ESTRANGEIRA	INGLÊS TÉCNICO	2	2		4	80
INFORMÁTICA	PROGRAMAS APLICATIVOS	4			4	80
	TÓPICOS ESPECIAIS			2	2	40
PROCESSAMENTO DE DADOS	FUNDAMENTOS EM INFORMÁTICA	3			3	60
	BANCO DE DADOS		2	5	7	140
	PROGRAMAÇÃO	8	6	6	20	400
MICRO - INFORMÁTICA	SISTEMAS OPERACIONAIS	3	3		6	120
	ARQUITETURA DE COMPUTADORES		2		2	40
	TECNOLOGIA DE HARDWARE	3			3	60
TELEMÁTICA	REDES DE COMPUTADORES		2	4	6	120
	TELAMÁTICA		2	3	5	100
COMPUTAÇAO GRÁFICA	MULTIMÍDIA			2	2	40
	EDITORAÇÃO ELETRÔNICA		3		3	60
ORGANIZAÇÃO DE EMPRESAS	GESTÃO EM INFORMÁTICA		3	3	6	120
ESTATÍSTICA	ESTATÍSTICA	2			2	40
SUBTOTAL		25	25	25	75	1500
ESTÁGIO						600
TOTAL						2100

Tabela 6.1: Grade Curricular do Curso Técnico de Informática.
Fonte: Núcleo de Informática e Sistemas

Quanto a estrutura física, o NIS possui seis laboratórios de ensino e pesquisa, atendendo uma demanda de 50 alunos do Curso de Informática e aproximadamente 800 alunos dos demais cursos.

A análise da grade curricular permite constatar a importância atribuída a disciplina PROGRAMAÇÃO na formação do técnico, já que corresponde a aproximadamente 27% da carga horária total do curso, excluído o estágio profissional.

Relativamente ao primeiro semestre, das 25 (vinte e cinco) aulas semanais, 8 (oito) são dedicadas aos conteúdos de PROGRAMAÇÃO, levando o aluno que inicia o curso a empregar grande parte de seu tempo em uma atividade que, na maioria das vezes, lhe é totalmente nova.

A superação da forma tradicional do ensino de lógica e linguagem de programação nos seus aspectos mais elementares é fundamental para o sucesso do curso como um todo.

Vislumbra-se a curto prazo, a proliferação e a utilização deste ambiente fora do contexto da aula presencial, permitindo aos alunos usufruir os benefícios, facilidades e potencialidades de acordo com seu interesse e disponibilidade de tempo.

6.3 - Aplicação

O processo de aplicação consiste numa etapa onde se pretende avaliar o grau de funcionalidade e importância que o ABC-Pro tende a proporcionar aos alunos do Curso Técnico de Informática.

A pesquisa por ser quantitativa não tende a avaliar a eficiência que o ambiente proporciona, mas sim, verificar a aceitabilidade dos alunos frente à uma nova ferramenta de programação.

Buscou-se expor o ABC-Pro nas turmas de 1ª e 2ª fase do Curso Técnico de Informática para testar seu funcionamento e o grau de adaptação. Além disso, avaliar a forma quanto ao tratamento e a complexidade de possíveis erros, bem como, testar e avaliar o grau de clareza no sistema de ajuda. O questionário de aplicação encontra-se em anexo.

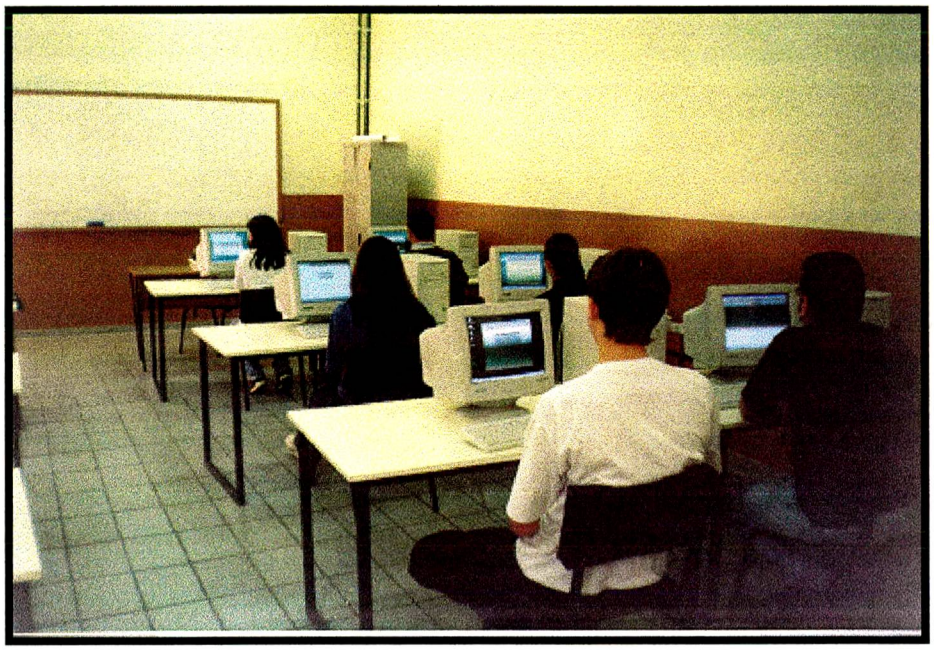


Foto 6.1 : Alunos do Curso Técnico de Informática utilizando o ABC-Pro.

Inicialmente, uma consideração quanto ao nível da clientela selecionada para o Curso Técnico de Informática. Por ser um curso pós segundo grau, a estimativa é que a procura maior seria de pessoas que não conseguiram ingressar numa faculdade ou que estão de uma maneira ou de outra atuando sem certificação na área buscando qualificação técnica profissional além das pessoas que gostam ou vislumbram a informática como tendência ou vocação promissora. A cada semestre, verifica-se que a procura de alunos graduandos ou recém-graduandos aumenta. Sem nos ater as causas ou implicações quanto ao futuro profissional que será exercido por eles, consideramos objetivamente que a qualificação técnica é acentuada. O índice que varia de 14 à 18 candidatos por vaga nos exames de seleção, faz com que "universitários" tenham mais facilidades de aprovação. Hoje, como mostra o gráfico 6.1, 47% dos alunos são graduandos. Além disso, os alunos que ingressaram no curso no último semestre possuem idade média de 20 anos, e 88% são do sexo masculino.

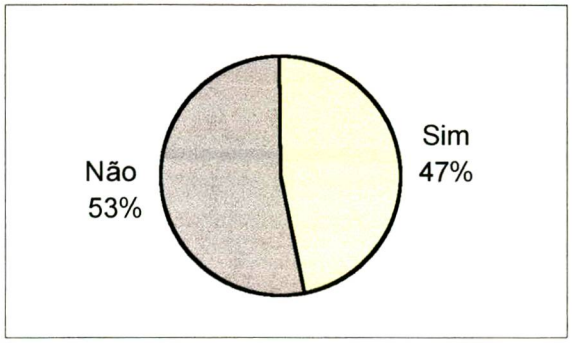


Gráfico 6.1: Percentual de Universitários.

Com relação à interface, a questão que envolve a disposição do menu de aplicação e a barra de ferramentas tiveram aceitabilidade de 100%. Os recursos que o menu oferece e a disposição da barra de ferramentas satisfazem pela simplicidade e objetividade. Na interface avaliada não havia-se implementado a utilização do agente. A tendência é que a interatividade com agentes melhore a aplicação tanto no aspecto visual quanto funcional.

Outra questão envolvendo a interface diz respeito à clareza com que os erros tratados são apresentados. Como resultado obteve-se 85% de clareza suficiente para a resolução de erros. É importante considerar a complexidade do tratamento de erros que deverá ser desenvolvido. Análises mais profundas devem ser estruturadas para que a mensagem de erro e suas sugestões descrevam adequadamente cada possibilidade. Em certos pontos, as mensagens não esclarecem com precisão o erro cometido. Natural e esperado por se tratar de um protótipo.

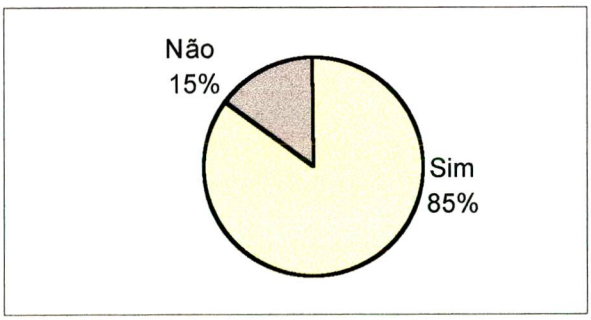


Gráfico 6.2 : Clareza na mensagens erro para sua resolução.

A utilização do Help, implica em duas situações possíveis:

1. O esclarecimento detalhado quanto à ocorrência de um erro (Help On-line);
2. Abordagem completa sobre o conteúdo da disciplina (Guia do Aprendizado).

Neste contexto, busca-se avaliar o grau de utilização, bem como suas principais necessidades. 93% utilizaram o Help como mostra o gráfico.

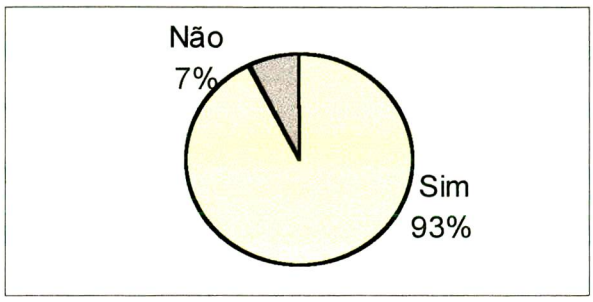


Gráfico 6.3 : Utilizaram o Help.

Sobre o uso do Help, daqueles que o utilizaram, 78% precisaram utilizar para ambas ou uma das situações em que estão estabelecidas.

Abaixo tem-se um gráfico que reflete a satisfação quanto a forma estrutural e o conteúdo do Help.

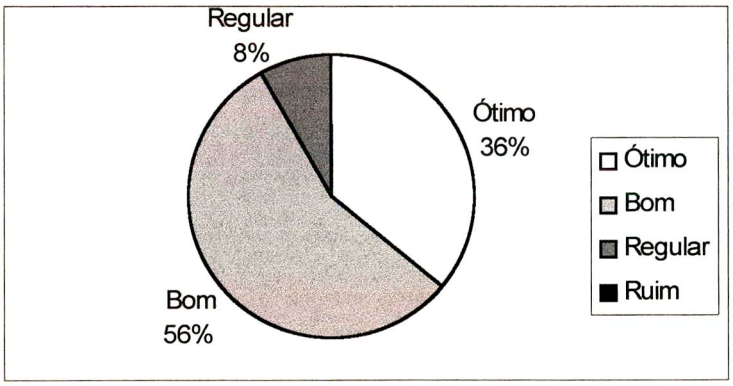


Gráfico 6.4 : Sobre a utilização do Help.

Como se trata de um protótipo, não aprofunda-se em detalhes complementares de alguns recursos. Deve-se reavaliar a estrutura do Help no sentido de proporcionar alternativas mais esclarecedoras como: maior quantidade de exemplos, uma estruturação dos temas mais adequadas e fácil, etc. O Help on-line utilizado para esclarecimento de erros do programa está restrito a uma dezena de erros.

Como destaque, e já esperado, o ponto forte considerado com maior evidência é justamente o fato da descrição do programa ser na língua que lhes é familiar. Instruções em português permitem maior entendimento e mais facilidade para

expor as instruções. As mensagens e a interface aparecem com 20% cada, como mostra o gráfico 6.5.

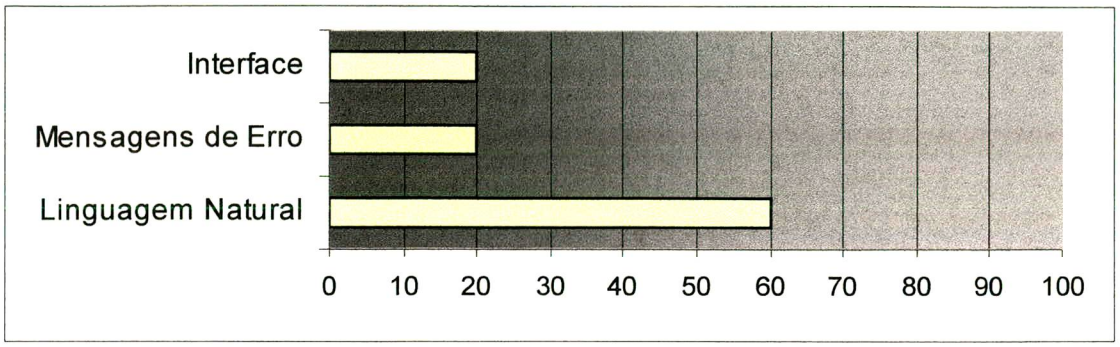


Gráfico 6.5 : Ponto forte do ABC-Pro.

Todo trabalho tende a repercutir num resultado. Este resultado, consequentemente mais esperado pelos alunos é, sem dúvida, a execução do programa desenvolvido possibilitando uma suposta satisfação quanto ao problema desenvolvido.

A execução do programa é o objetivo final de qualquer programador. É a forma mais cômoda de avaliar logicamente se o programa está correto ou não. Como todos já estavam utilizando alguma linguagem de programação como o pascal, a execução do programa como maior necessidade já era esperado. Este índice alcançou 40% dos alunos na identificação do ponto fraco do trabalho seguido do sistema de ajuda com 20% e alguns detalhes de implementação com 13%.

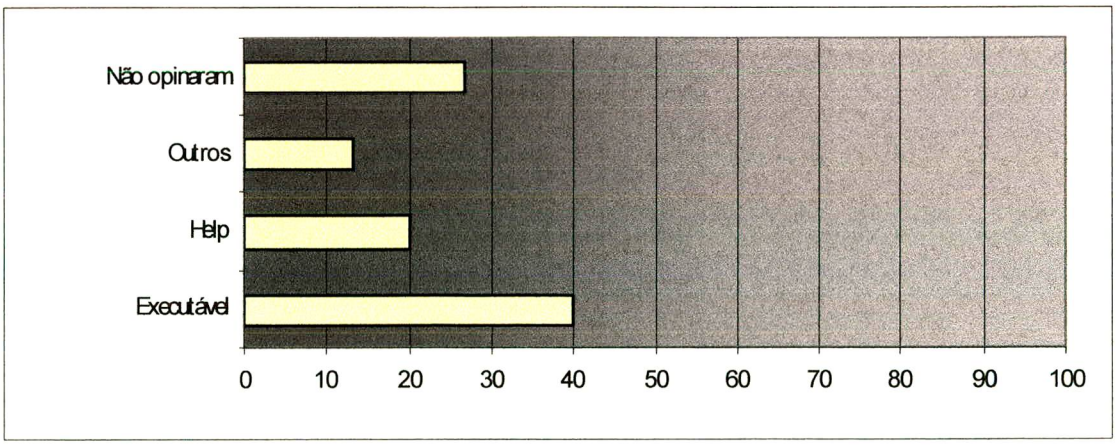


Gráfico 6.6 : Ponto fraco do ABC-Pro.

Consideraram a importância da disposição do ambiente como processo inicial de aprendizagem já que, para eles, a dificuldade natural (em função do

processo pedagógico) que tiveram para desenvolvimento de algoritmos poderia ser extremamente reduzida. A sugestão foi unânime.

A avaliação frente a utilização do ABC-Pro pelos alunos do Curso Técnico de Informática repercutiu nos seguintes aspectos:

- o pensamento lógico é descrito com muito mais facilidade pois a linguagem (como se fosse um algoritmo (portugol)) não oferece grandes restrições;
- um algoritmo que é exposto à análise de um compilador;
- uma estrutura completa (opcional) de mensagens de erros que permite sua identificação, a linha de ocorrência e, inclusive, sugestões para sua resolução;
- auxílio na compreensão das deficiências das suas soluções através do help on-line;
- o conteúdo da disciplina exposta num "guia", bem como a orientação de sua forma gramatical;
- o desafio proporcionado pela omissão de uma ou mais informações nas situações que ocasionam as mensagens de erro;
- a facilidade proporcionada pelo ambiente pelas características que tem em comum com outros *softwares* (salvar, abrir, novo, copiar, recortar, colar, fonte, ajuda, entre outros);

Uma tendência que já é uma realidade, é desenvolvimento em questões complementares de conteúdos envolvidos no processo de ensino de programação. A utilização de vetores/matrizes, funções e procedimentos deverão, em breve, fazer parte da gramática estabelecida.

O planejamento está bem determinado, através da aplicação na realidade educacional obteve-se fundamentos para a viabilização do ABC-Pro como ferramenta na disciplina de Programação no primeiro semestre do curso. Pode-se, dando continuidade ao projeto, complementar a implementação, certos de que a aceitabilidade e o recurso técnico oferecido para os iniciantes da programação estará dentro dos padrões pedagógicos. Isto permitirá ao aluno, a aplicação de seus conhecimentos de forma prática. Esta atividade tende a uma produção qualitativa e quantitativa do conhecimento se comparado com os

métodos tradicionais. A avaliação quanto aos aspectos gerais do ABC-Pro é mostrada no gráfico 6.7.

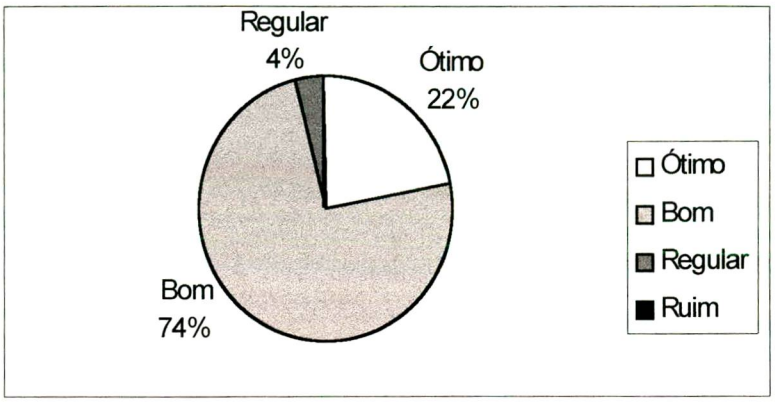


Gráfico 6.7: Utilização do ABC-Pro.

Para ilustrar o trabalho realizado, abaixo tem-se a citação de um dos alunos que utilizaram o ABC-Pro refletindo uma situação em que muitos alunos se deparam.

> "Uma de minhas grandes dificuldades no princípio foi com programação. Em geral desenvolvemos o algoritmo para a solução de um problema primeiro no papel para depois implementarmos na máquina. E aí surgem os problemas, pois os erros só aparecem neste momento e nos tomam muito tempo para conserta-los. Excetuando os problemas de lógica, os erros são, em geral, nas estruturas e na linguagem pois no papel não temos nada que nos corrija. (...) O ABC-Pro permite escrever um algoritmo em linguagem de alto nível e realizar as correções, isso diminui as chances de erro dando mais tempo para nos preocupar com a lógica do problema." Edivar Brustolin (aluno do Curso Técnico de Informática)

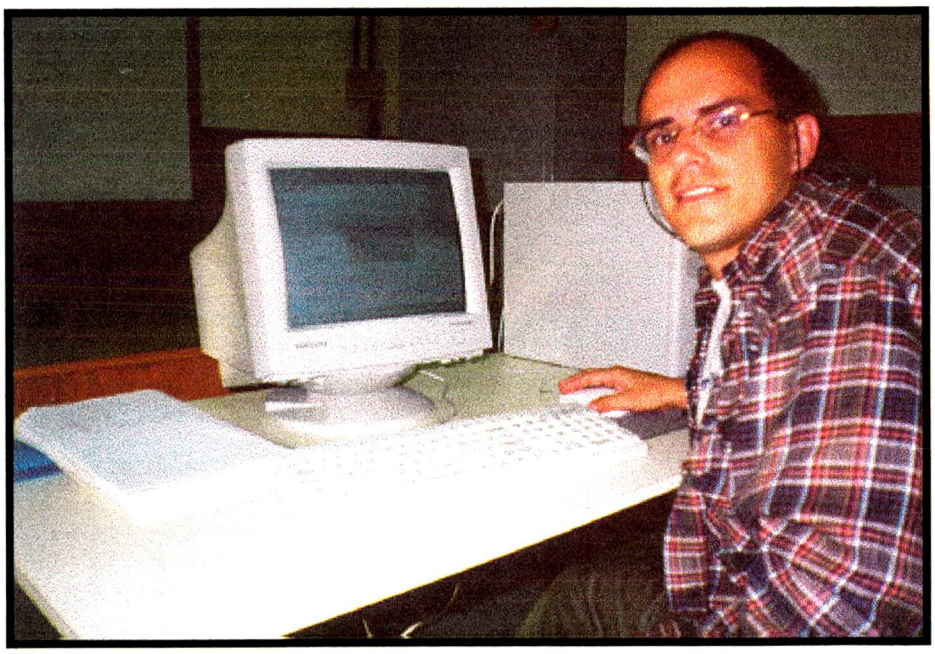


Foto 6.2 : Aluno usando o ABC-Pro.

Neste contexto, Vigostsky (1991, p.62) coloca que:

> "Quando o homem introduz uma modificação no ambiente através de seu próprio comportamento, essa modificação vai influenciar seu comportamento futuro."

Este tipo de processo de captação de informação percebido, que envolve atenção ao significado, tem influência positiva na lembrança. O que o aluno faz e o que ele aprende estão relacionados.

A finalidade do ABC-Pro não é apenas minimizar o percentual de reprovações/desistências em cursos que trabalham com o aprendizado de lógica de programação, mas, principalmente, maximizar a qualidade do conhecimento do aluno nesta área.

CAPÍTULO VII

7 - CONCLUSÃO E PROPOSTAS FUTURAS

7.1 - Conclusões Gerais

Conhecer é sempre uma ação que demanda esquemas de assimilação e acomodação num processo constante de reorganização que é fruto da atividade daquele que interage com o mundo.

Este trabalho busca uma alternativa para beneficiar o processo ensino-aprendizagem da lógica de programação através do desenvolvimento de uma ferramenta computacional. Tentou-se proporcionar através deste trabalho um ambiente de fácil utilidade com suporte à desenvolvimentos de programas de forma clara e idêntica à algoritmos.

Na aplicação com alunos do Curso Técnico de Informática do CEFET/SC, constatou-se que o processo de desenvolvimento do raciocínio lógico pode evoluir pois o ambiente proporciona processos nos quais o aluno constrói seu conhecimento.

A facilidade para desenvolvimento de programas é o fator preponderante no ABC-Pro pois o aluno escreve um programa com a mesma clareza com que escreve um algoritmo. Para avaliar este programa, o processo de compilação realiza a análise (léxica, sintática e semântica).

O conhecimento para construir o compilador, núcleo do trabalho, foi efetivamente o requisito técnico fundamental para o sucesso do funcionamento do ABC-Pro. Considera-se, ainda, o progresso para complementar esta ferramenta com a implementação da fase de síntese (gerador de código intermediário, otimizador e código final) gerando um arquivo executável.

Integrado ao processo de compilação, implementou-se um sistema de ajuda com o objetivo de apoiar o aluno na resolução de problemas e ter à disposição uma forma de consulta sobre o conteúdo de programação. Como no processo construtivista, considera-se o erro como fator de aprendizagem, pois faz com que o aluno busque uma nova solução para o problema. Neste processo, o benefício proporcionado pelo sistema de tratamento de erros garante esclarecimento do problema, além da exploração do conteúdo em questão. Com a identificação da origem do erro, usando o erro como fator construtivo, o aluno estabelece o melhor de suas ações.

Cabe ao professor, o auxílio aos alunos no que diz respeito à sintaxe de ABC-Pro. Apesar de possuir uma sintaxe fácil e clara, como uma forma de comunicação usual, tem-se que garantir o entendimento de acordo com um vocabulário e a gramática estabelecida.

Não pode-se desconsiderar a experiência do professor. As técnicas e processos de raciocínio lógico devem ser permanentemente repassadas e exploradas para que análises e críticas proporcionem um método (não o único) adequado para cada problema. Através disso, possibilita-se a descoberta de diferente caminhos na resolução de problemas.

O aluno aprende por exploração, investigação e descoberta contribuindo para o desenvolvimento de suas estruturas cognitivas. Com isso, um esquema retido pode ser transferido com muito mais facilidade.

O aluno adquirirá uma nova compreensão do significado e da utilidade daquilo que faz, do seu "fazer", e como é o agente criador e transformador do próprio conhecimento, pode-se dizer que ele constrói o seu "saber". Se o professor

possuir a consciência desse referencial pedagógico e permitir que este norteie o "seu fazer", estará adquirindo uma nova postura, que garantirá o aparecimento também de um novo tipo de aluno: conhecedor de sua realidade, criativo, mais crítico e participativo.

O paradigma das idéias pedagógicas tem a educação como um processo de desenvolvimento do ser humano. Na filosofia construtivista, as condições adequadas desse desenvolvimento permitem emergir no aluno suas reais potencialidades.

Buscou-se através da pesquisa, bases para construir o ambiente determinado por critérios pedagógicos que possibilitasse a construção do conhecimento. Para o professor, uma forma para proporcionar desafios que motivem a descoberta através de uma ferramenta computacional determinando assim, que o processo eficiente de assimilação e acomodação.

Finalmente, a disponibilização do ambiente em uma plataforma que suporte o ensino à distância, através da Internet, deve ser uma meta de curto prazo, pelo destaque que esta mídia vem tendo como instrumento de ensino e aprendizagem.

Esta foi a contribuição desenvolvida para educadores que buscam ferramentas e metodologias adequadas no processo ensino-aprendizagem da lógica de programação. Através desta contribuição, disponibiliza-se um instrumento integrado aos propósitos educacionais. Considerando-se no emprego dessa tecnologia o seguinte alerta, apresentado por Alencar (1995, p.85):

> "... na medida em que a escola contribuir para formar no aluno o pensamento crítico e criador e se preocupar não apenas com a capacidade do aluno reproduzir informações, mas também de produzir conhecimento, ela estará dando sua parcela de contribuição para que ultrapassemos alguns dos problemas com os quais convivemos no momento e para que nos habilitemos a enfrentar, de forma mais adequada, problemas futuros."

7.2 – Propostas Futuras

Como propostas futuras tem-se além do aperfeiçoamento da ferramenta, a utilização de recursos tecnológicos alternativos melhorando e diversificando o processo pedagógico do ensino da lógica de programação.

O objetivo final da ferramenta ABC-Pro é a implementação completa que consiste em: aperfeiçoamento da interface; aprimorar e complementar as atividades de ajuda; gerar código executável; proporcionar *debug* para acompanhar o funcionamento de programas passo a passo; disponibilização do ambiente num sistema distribuído permitindo ao mediador o controle das mensagens de erro que podem ser utilizadas pelos alunos; complementar a gramática proposta incluindo a utilização de *array's*, procedimentos e funções, entre outros; disponibilizar ferramentas no ambiente como: endentação automática, processo *highlighting* (destacar, por exemplo palavras reservadas, comentários, variáveis, etc.) e gerador de fluxogramas para proporcionar graficamente a visualização da seqüência das instruções.

O desenvolvimento de um ambiente hipermídia para o aprendizado da lógica de programação.

A hipermídia hoje, com os recursos advindos do avanço tecnológico, permite a comunicação entre o homem e o computador com fluxo de informações nos dois sentidos, tornando muito mais lúdico e rápido o processo de busca de informações e aprendizagem, unindo recursos de imagem, som, vídeo, texto, etc. Este recurso permite a educação de estudantes conseguindo sua atenção, fornecendo informações, favorecendo a discussão através da interatividade para assimilar e refletir o conteúdo. Estes fatores contribuem para a motivação do estudante para que ocorra o processo de ensino-aprendizagem.

Neste sistema, o aluno aprenderá as diversas e variadas formas de composição de programas de forma modularizada e interativa e, no decorrer, a elaboração por completo de programas. Isto permitirá ao aluno maior autonomia do aprendizado.

Um protótipo foi desenvolvido, com exceção do compilador, e serviu como conclusão da disciplina de Multimídia no Ensino, inclusive, com aplicação em alunos da 1ª fase do Curso Técnico de Informática tendo ótima aceitação.

Outra tendência natural é a integração do ABC-Pro com o Ambiente para Ensino da Lógica de Programação. Esta ferramenta serviu como dissertação de mestrado do professor Marco Neiva Koslosky (professor do CEFET/SC). O objetivo foi a implementação de um ambiente de aprendizagem baseado em casos para utilização na disciplina de Programação do Curso Técnico de Informática. O uso da metodologia de Raciocínio Baseado em Casos - RBC foi escolhido em razão da semelhança do seu ciclo de atividades e as etapas percorridas pelos alunos na resolução de problemas.

Uma das vantagens decorrentes deste ambiente foi a facilidade de representação do conhecimento. Utilizou-se de técnicas de banco de dados para armazenar e recuperar os casos e a capacidade do sistema em aprender pelo armazenamento de novos casos.

A integração destes trabalhos permitirá ao aluno o desenvolvimento de programas possibilitando a verificação lógica do mesmo, bem como a utilização de uma base de casos para validação de seus conhecimentos. Além disso, o acompanhamento do professor permitindo localizar as principais dificuldades pelos alunos na resolução de problemas.

Não pode-se, também, deixar de contemplar estudos de programação alternativos. Com base no trabalho desenvolvido é importante que as novas tendências no desenvolvimento de programação sejam estudadas no sentido de que ferramentas pedagógicas sejam viabilizadas para, pelo menos, diminuir as dificuldades no aprendizado.

A Programação Orientada à Objetos surge como tendência renovadora no processo de desenvolvimento de programas consistindo em conteúdo

obrigatório em diversas entidades que oferecem cursos técnicos e graduação. Neste sentido, pode-se estabelecer um ambiente integrado para aprendizagem de POO.

CAPÍTULO VIII

8 - BIBLIOGRAFIA

AHO, A.V., SETHI, R., ULLMAN, J.D. **Compiladores: Princípios, Técnicas e Ferramentas**. Rio de Janeiro. LTC, 1995.

ALENCAR, Eunice M.L. **Criatividade**. 2ª Edição. Brasília. Editora Universidade de Brasília, 1995.

ANDRADE, Adja F. **Abordagem Sistemática para Criação de Mundos Virtuais: Uma Ferramenta para Desenvolvimento da Criatividade**. Florianópolis: UFSC, 1998 (Trabalho Individual de Pós-Graduação em Ciências da Computação).

BOLZAN, Regina F. A. **O Conhecimento Tecnológico e o Paradigma Educacional**. Florianópolis: UFSC, 1998 (Dissertação de Mestrado em Engenharia de Produção).

BRUNER, Jerome S. **Uma Nova Teoria da Aprendizagem**. Rio de Janeiro. Bloch, 1969.

CABRERA, L. G., CABRERA, R., CEJUDO M. L. R. **Areflexionantes sobre el uso del ordenador en la educatión** - Revista Novática, Sep/oct, No 117, 1995.

CAMPOS, D.M.S. **Psicologia da aprendizagem**. Petrópolis. Vozes, 1972.

CARDOZO, Claudine M. e RAMOS, Edla M. F. **AALO : Um ambiente para a aprendizagem de lógica** - VI Simpósio Brasileiro de Informática na Educação. Florianópolis: UFSC, 1995.

CASAS, L. A. A. **Contribuições para a Modelagem de um Ambiente Inteligente de Educação Baseado em Realidade Virtual** - Tese de Doutorado em Engenharia de Produção, UFSC, Florianópolis, 1998.

CHAVES, Eduardo O. C. **Multimídia: Conceituação, Aplicações e Tecnologia**. São Paulo. People Computação, 1991.

DOCKTERMAN, David A. **Great Teaching in the One Computer Classroom**. Cambridge. Massachusetts. Tom Snyder Productions, 1991.

DUARTE, D. **Livro da Ensinança de Bem Cavalgar toda Sela**. Leal conselheiro - in: Obras dos Príncipes de Avis, Porto/Portugal:Lello, 1981.

FERREIRA, B.W., VERONESE, L.V., FARIA,E.T., MALLMANN, J., NUNES, M.L.T., ALVES,M.R.I.P., AVELINE,S. **Psicologia Pedagógica**. Porto Alegre. Sulina, 1985.

FORBELLONE, A.L.V., EBERSPÄCHER, H.F. **Lógica de Programação - A Construção de Algoritmos e Estruturas de Dados**. São Paulo. McGraw-Hill, 1993.

FREIRE, Paulo. **Papel da Educação na Humanização**. Rio de Janeiro. Paz e Terra, 1971.

FURTADO, Olinto J. V. **Linguagens Formais e Compiladores** (apostila). Florianópolis: UFSC, 1987.

GAGNÉ, Robert M. **Como se Realiza a Aprendizagem**. Rio de Janeiro – LTC, 1974.

GALLOWAY, Charles. **Psicologia da Aprendizagem e do Ensino**. São Paulo. Cultrix, 1981.

GARDNER, Howard. **Estruturas da Mente: a Teoria das Inteligências múltiplas**. Porto Alegre. Artes Médicas, 1994.

GARDNER, Howard. **Inteligências Múltiplas: a Teoria na Prática**. Artes Médicas, 1995.

GREENE, Judith. **Pensamento e Linguagem**. Rio de Janeiro. Zahar, 1976.

GUIMARÃES, A.M., LAGES, N.A. **Algoritmos e Estruturas de Dados**. Rio de Janeiro. LTC, 1985.

HAWKINS, J. **O uso de novas Tecnologias na Educação**. Rio de Janeiro. Revista TB, vol 120, Jan/Mar 1995.

HILL, Winfred F. **Aprendizagem**. Rio de Janeiro. Guanabara Dois, 1981.

Instituto Brasileiro de Pesquisa em Informática. **Técnicas de Programação com Pascal**. Rio de Janeiro. IBPI, 1993.

KELLER, Fred S. **Aprendizagem: Teoria do Reforço**. São Paulo. Ed. Pedagógica e Universitária, 1973.

HOWE, Michael J. A. **Introdução à Psicologia da Aprendizagem**. São Paulo. Vértice, 1986.

KOSLOSKY, MARCO A. N. **Aprendizagem Baseada em Casos: Um Ambiente para Ensino de Lógica de Programação**. Florianópolis: UFSC, 1999 (Dissertação de Mestrado em Engenharia de Produção).

LEIF, J. **Vocabulário técnico e crítico da pedagogia e das ciências da Educação**. Lisboa/Portugal. Notícias, 1976.

LUCKESI, Cipriano C. **Filosofia da Educação**. São Paulo. Cortez, 1991.

LURIA, Alexander R. **Desenvolvimento Cognitivo**. São Paulo. Ícone, 1990.

MENDES, Mônica H. **A Informática na Escola**. Jornal Psicopedagogia. Goiânia. Ano I, n. 2, maio/jun. 1995.

MORAN, José M. **Artigo: A Escola do Amanhã: Desafio do Presente**. 1993.

MORENTE, M.G. **Fundamentos de Filosofia - Lições preliminares**. 2ª edição. São Paulo. Mestre Jou, 1966.

NUNES, Rosemeri C. **Metodologia para o Ensino de Informática para a Terceira Idade - Aplicação no CEFET/SC**. Florianópolis: UFSC, 1999 (Dissertação de Mestrado em Engenharia de Produção).

OLIVEIRA, Marta K. **Vygotsky: Aprendizado e desenvolvimento, um processo sócio-histórico**. São Paulo. Scipione, 1995.

PAPERT, S. **Logo: Computadores e Educação**. São Paulo. Editora Brasiliense, 1985.

PIAGET, Jean. **A Equilibração das Estruturas Cognitivas**. Rio de Janeiro. Zahar, 1976.

PIAGET, Jean. **Psicologia e Pedagogia**. São Paulo. Forense, 1970.

PIAGET, Jean. **Psicologia da Inteligência**. São Paulo. Fundo de Cultura, 1967.

PIAGET, Jean. **O Nascimento da Inteligência na Criança**. Rio de Janeiro. Guanabara, 1987.

Que Development Group. **Introdução à Programação**. Rio de Janeiro. Campus, 1993.

REINHARDT, Andy. **Novas Formas de Aprender** - Byte Brasil. São Paulo, v. 4, n. 3, mar. 1995.

SALIBA, Walter L. C. **Técnicas de Programação - Uma Abordagem Estruturada**. São Paulo. McGraw-Hill, 1992.

SERPA, Maria da Glória N. **O Impacto da Informática na Educação: o caso do Distrito Federal**. Dissertação (Mestrado em Educação) - Faculdade de Educação, Universidade de Brasília, 1986.

SETZER, V.W., MELO, I.S.H. **A Construção de um Compilador**. Rio de Janeiro. LTC, 1983.

SEVERINO, Antônio J. **Metodologia do Trabalho Científico**. São Paulo. Cortez, 1996.

SILVA, J. C. G. e ASSIS, F. S. G. **Linguagens de Programação - Conceitos e Avaliação**. Mc Graw-Hill, 1988.

SOARES, Felipe C. **Otimização do Ensino de Informática através da Aplicação dos Conceitos de Ergonomia no Ambiente Físico. Um Estudo de Caso: Curso Técnico de Informática do CEFET/SC**. Florianópolis: UFSC, 1999 (Dissertação de mestrado em Engenharia de Produção).

TREMBLAY, J. P. e SORENSON, P. G. **The Theory and Practice of Compiler Writing**. Mc Graw-Hill, 1985.

ULBRICHT, Vânia R. **Modelagem de um Ambiente Hipermídia de Construção do Conhecimento em Geometria Descritiva**. Florianópolis: UFSC, 1997 (Tese de doutorado em Engenharia de Produção).

VALENTE, J. A. **O Professor no Ambiente Logo: formação e atuação**. Campinas. Gráfica da UNICAMP, 1996.

VILARINHO, Lúcia R. G. **Didática: Temas Selecionados**. Rio de Janeiro. LTC, 1986.

VYGOTSKY, L. S. **A Formação Social da Mente : O Desenvolvimento dos Processos Psicológicos Superiores**. São Paulo. Martins Fontes, 1991.

WECHSLER, S. M. **Criatividade: Descobrindo e Encorajando**. Campinas. Ed. Psy, 1993.

HTTP://WWW.MUSEUDOCOMPUTADOR.COM.BR/

HTTP://HISTORIACOMP.VIRTUALAVE.NET/

HTTP://MEMBERS.TRIPOD.COM/LFCAMARA/

ANEXOS

ANEXO I

RELAÇÃO DE TOKENS UTILIZADOS NO ABC-Pro

RELAÇÃO DE TOKENS

Item Léxico	token	Classe
programa	1	palavra reservada
inicio	2	palavra reservada
fim	3	palavra reservada
var	4	palavra reservada
caracter	5	palavra reservada
inteiro	6	palavra reservada
real	7	palavra reservada
booleano	8	palavra reservada
leia	9	palavra reservada
escreva	10	palavra reservada
verdadeiro	11	palavra reservada
falso	12	palavra reservada
constante	13	palavra reservada
literal[1]	14	literal
ou	15	palavra reservada
e	16	palavra reservada
nao	17	palavra reservada
se	18	palavra reservada
entao	19	palavra reservada
senao	20	palavra reservada
fimse	21	palavra reservada
escolha	22	palavra reservada
fimescolha	23	palavra reservada
enquanto	24	palavra reservada
faca	25	palavra reservada
fimenquanto	26	palavra reservada
para	27	palavra reservada
de	28	palavra reservada
ate	29	palavra reservada
fimpara	30	palavra reservada
identificador[2]	31	identificador
número inteiro[3]	32	constante
número real[4]	33	constante
+	34	símbolo especial simples
-	35	símbolo especial simples
*	36	símbolo especial simples
/	37	símbolo especial simples
(	38	símbolo especial simples
)	39	símbolo especial simples
:	40	símbolo especial simples
=	41	símbolo especial simples

[1] Ver figura 5.4
[2] Ver figura 5.2
[3] Ver figura 5.3
[4] Ver figura 5.3

>	42	símbolo especial simples
<	43	símbolo especial simples
,	44	símbolo especial simples
;	45	símbolo especial simples
>=	46	símbolo especial duplo
<=	47	símbolo especial duplo
<>	48	símbolo especial duplo

ANEXO II

GRAMÁTICA: ESTRUTURA SINTÁTICA DO ABC-Pro

GRAMATICA ABC_Pro

Símbolos Não-Terminais

Vn = { <programa> <BLOCO> <LISTA_DECL> <DECL_CONST> <LD_CONST>
<LISTA_ID> <NOVO_ID> <TIPO_CONST> <DECL_VAR> <LD_VAR>
<TIPO> <CORPO> <LISTA_COMANDOS> <COMANDO> <SENAOPARTE>
<EXPRESSAO> <EXP_SIMP> <REPEXPSIMP> <TERMO> <REP_EXPR>
<FATOR> <REPTERMO> <OPMULT> <OPADD> <RESTO_EXP> <OPREL>
<LISTA_CASOS> <CONSTANTE> <REPLCONST> <REP_CASOS>
}

Símbolos Terminais

Vt = {	programa	inicio	fim	var
	caracter	inteiro	real	booleano
	leia	escreva		
	verdadeiro	falso	constante	literal
	ou	e	nao	
	se	entao	senao	fimse
	escolha	fimescolha		
	enquanto	faca	fimenquanto	
	para	de	ate	fimpara
	id	num_inteiro	num_real	
	'+'	'-'	'*'	'/'
	'('	')'	':'	'='
	'>'	'<'	','	';'
	'>='	'<='	'<>'	}

Produções

S = <programa>

P = {

<programa> ::= programa id #1 ';' <BLOCO> ;

<BLOCO> ::= <LISTA_DECL> <CORPO> ;

<LISTA_DECL> ::= <DECL_CONST> <DECL_VAR> ;

<DECL_CONST> ::= constante #2 <LISTA_ID> #3 '=' <TIPO_CONST> #4 ';'
<LD_CONST> | î;

<LD_CONST> ::= #2 <LISTA_ID> #3 '=' <TIPO_CONST> #4 ';'
<LD_CONST> | î ;

<LISTA_ID> ::= id #5 <NOVO_ID> ;

<NOVO_ID> ::= ',' id #5 <NOVO_ID> | î ;

```
<TIPO_CONST>    ::= num_inteiro #11 | num_real #12 | verdadeiro #13 | falso
                    #14 | literal #15;

<DECL_VAR>      ::= var #2 <LISTA_ID> #3 ':' <TIPO> #6 ';' <LD_VAR>  | î;

<LD_VAR>        ::= #2 <LISTA_ID> #3 ':' <TIPO> #6 ';' <LD_VAR> | î ;

<TIPO>          ::= inteiro #7 | real #8 | booleano #9 | caracter #10 ;

<CORPO>         ::= inicio <LISTA_COMANDOS> fim | î ;

<LISTA_COMANDOS> ::= <COMANDO> ';' <LISTA_COMANDOS> | î ;

<COMANDO>       ::= id #16 '=' <EXPRESSAO> #17 |
                    leia id #18 |
                    escreva <EXPRESSAO> #41 <REP_EXPR> #41 |
                    se <EXPRESSAO> #19 entao <LISTA_COMANDOS>
                            <SENAOPARTE> fimse |
                    escolha <EXPRESSAO> de <LISTA_CASOS> fimescolha |
                    para id #20 de <EXPRESSAO> #21 ate <EXPRESSAO> #22
                            faca <LISTA_COMANDOS> fimpara |
                    enquanto <EXPRESSAO> #19 faca <LISTA_COMANDOS>
                            fimenquanto ;

<LISTA_CASOS>   ::= <CONSTANTE> <REPLCONST> ':' <LISTA_COMANDOS>
                    <REP_CASOS> ;

<CONSTANTE>     ::= id | <TIPO_CONST> ;

<REPLCONST>     ::= ',' <CONSTANTE> <REPLCONST> | î ;

<REP_CASOS>     ::= ';' <LISTA_CASOS> | senao <LISTA_COMANDOS> | î ;

<REP_EXPR>      ::= ',' <EXPRESSAO> <REP_EXPR> | î ;

<SENAOPARTE>    ::= senao <LISTA_COMANDOS> | î ;

<EXPRESSAO>     ::= <EXP_SIMP> <RESTO_EXP> ;

<EXP_SIMP>      ::= <TERMO> <REPEXPSIMP> ;

<REPEXPSIMP>    ::= <OPADD> <TERMO> <REPEXPSIMP> | î ;

<TERMO>         ::= <FATOR> <REPTERMO> ;

<FATOR>         ::= nao <FATOR> #38 | '(' #39 <EXPRESSAO> #40 ')' | id #23 |
                    <TIPO_CONST> #24;

<REPTERMO>      ::= <OPMULT> <FATOR> <REPTERMO> | î ;
```

```
<OPMULT>        ::= '*' #28 | '/' #29 | e #30 ;

<OPADD>         ::= '+' #25 | '-' #26 | ou #27 ;

<RESTO_EXP>     ::= <OPREL> #37 <EXP_SIMP> | î ;

<OPREL>         ::= '=' #31 | '<' #32 | '>' #33 | '>=' #34 | '<=' #35 | '<>' #36;
      }
```

ANEXO III

ALGORITMO DAS AÇÕES SEMÂNTICAS DO ABC-Pro

Algoritmos de algumas Ações Semânticas

Ação 1 (#1)
Inserir identificador na Tabela de Símbolo (TS)
categoria = 'nome do programa'

Ex.:
PROGRAMA TESTE;
CONSTANTE A,B = 10;
VAR X : CARACTER;

TABELA DE SÍMBOLOS

Nome_id	Categoria	Valor	Tipo
TESTE	nome do programa		
A	constante	10	inteiro
B	constante	10	inteiro
X	variável		caracter

Ação 2 (#2)
Gerar posição da TS
início_lista = índice da TS

Ação 3 (#3)
Gerar posição da TS
fim_lista = índice da TS

Ação 4 (#4)
Gerar constantes na TS
Para i=início_lista até fim_lista
    TS[i].categoria = 'constante'
    TS[i].valor = <constante>
    TS[i].tipo = tipo_const

Ação 5 (#5)
Verificar se identificador já está na TS
se identificador pertence TS
    se categoria_identificador = 'nome do programa'
        ERRO('nome do programa não pode ser utilizado')
    senão
        ERRO('identificador já declarado')
senão
    inserir identificador na TS

Ação 6 (#6)
Gerar variáveis na TS
Para i=início_lista até fim_lista
    TS[i].categoria = 'variável'
    TS[i].tipo = tipo_atual (real, inteiro, caracter, booleano)

-
-
-

Ação 16 (#16)
Verificar se identificador está na TS
se identificador não pertence TS
    ERRO('identificador não foi declarado')
senão
    Guardar posição => posição = índice da TS

Ação 17 (#17)
Verificar se tipo do resultado da expressão = tipo_identificador
se tipo_identificador => TS[posição].tipo <> tipo da expressão
    se TS[posição].tipo <> real ou tipo_expressão <> inteiro {real aceita
        ERRO('tipos incompatíveis') inteiro}

Ação 18 (#18)
Verificar se identificador já foi declarado e se é variável
se id não pertence TS
    ERRO('identificador não foi declarado')
senão
    se categoria_identificador <> 'variável'
        ERRO('identificador não é variável')

-
-
-

ANEXO IV

QUESTIONÁRIO APLICADO NO CURSO TÉCNICO DE INFORMÁTICA

UNIVERSIDADE FEDERAL DE SANTA CATARINA
DEPARTAMENTO DE ENGENHARIA DE PRODUÇÀO E SISTEMAS
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA DE PRODUÇÃO

Florianópolis, abril de 2000.

QUESTIONÁRIO DE APLICAÇÃO SOBRE O ABC-Pro

Dados Pessoais

Nome:_____ Fase: ___ Idade: ___ Sexo: ___
(o preenchimento do nome é opcional)

Formação (2º Grau) Técnica?

☐ Sim, qual? _____

☐ Não

Está fazendo (ou fez) curso superior?

☐ Sim, qual? _____

☐ Não

1- Antes desta disciplina, você já havia desenvolvido programas utilizando alguma linguagem de programação?

☐ Sim, qual? _____

☐ Não

2- Sobre a interface responda:

2.1- O menu e a barra de ferramentas satisfazem as condições básicas quanto a utilização?

☐ Sim

☐ Não, especifique sugestões/melhorias

2.2- As mensagens de erro foram claras o suficiente para que você resolvesse os problemas do algoritmo?

☐ Sim

☐ Não, por que?

2.3- Você utilizou o Help?

☐ Sim: ☐ Não

Foi porque:

- ☐ Teve curiosidade
- ☐ Precisou resolver um erro do programa
- ☐ Precisou consultar o tutorial para questões complementares de aprendizagem

Sobre a utilização do Help, você considerou:

- ☐ Ótimo
- ☐ Bom
- ☐ Regular
- ☐ Ruim

3- Facilitou o entendimento para utilização de uma outra linguagem de programação?

☐ Sim

☐ Não, por que?

4- Qual o ponto forte do ambiente?

5- Qual o ponto fraco do ambiente?

6- Considera-se mais seguro para desenvolver programas após utilizar o ambiente?

☐ Sim

☐ Não, por que?

7- Sobre a utilização do ABC-Pro de maneira geral, você considerou:

- ☐ Ótimo
- ☐ Bom
- ☐ Regular
- ☐ Ruim

8- Para melhorar o ambiente, o que seria necessário?