UNIVERSIDADE REGIONAL DE BLUMENAU

CENTRO DE CIÊNCIAS EXATAS E NATURAIS

CURSO DE CIÊNCIAS DA COMPUTAÇÃO – BACHARELADO

FERRAMENTA PARA APOIO AO ENSINO DE INTRODUÇÃO À PROGRAMAÇÃO

KARLY SCHUBERT VARGAS

BLUMENAU
2005

2005/1-32

KARLY SCHUBERT VARGAS

FERRAMENTA PARA APOIO AO ENSINO DE INTRODUÇÃO À PROGRAMAÇÃO

Trabalho de Conclusão de Curso submetido à Universidade Regional de Blumenau para a obtenção dos créditos na disciplina Trabalho de Conclusão de Curso II do curso de Ciências da Computação — Bacharelado.

Prof. Joyce Martins , MsC. - Orientadora

BLUMENAU
2005

2005/1-32

FERRAMENTA PARA APOIO AO ENSINO DE INTRODUÇÃO À PROGRAMAÇÃO

Por

KARLY SCHUBERT VARGAS

Trabalho aprovado para obtenção dos créditos na disciplina de Trabalho de Conclusão de Curso II, pela banca examinadora formada por:

Presidente:	Prof. Joyce Martins, MsC. – Orientadora, FURB
Membro:	Prof. Jomi Fred Hubner, Dr. – FURB
Membro:	Prof. Paulo Roberto Dias, MsC. – FURB

Blumenau, 29 de junho de 2005

Dedico este trabalho a todas as pessoas que me ajudaram diretamente na realização deste, em especial a minha família pelo apoio em todos os momentos.

AGRADECIMENTOS

A Deus, pelo seu imenso amor e graça.

À minha família, por toda a base que me deram durante todos estes anos.

Aos meus amigos, pelos empurrões e cobranças. A ajuda de vocês foi essencial para que eu pudesse chegar até aqui.

Ao Rodrigo Hackbarth por sua ajuda no *abstract*.

Ao professor Paulo Dias, por toda a ajuda dada na definição da linguagem e nos testes da ferramenta, além de ter utilizado a ferramenta na disciplina.

À minha orientadora, Joyce Martins, por ter acreditado na idéia e ajudado na concretização da mesma.

Para finalizar, quero agradecer a todos os professores e colegas, que me acompanharam durante minha graduação, pelo conhecimento e experiências trocados durante o curso.

Não é digno de saborear o mel, aquele que se afasta da colméia com medo das picadelas das abelhas.

Willian Shakespeare

RESUMO

O presente trabalho tem por finalidade descrever o desenvolvimento de uma ferramenta para apoio ao ensino de introdução à programação, permitindo o desenvolvimento de algoritmos em uma linguagem de programação estruturada e em português. A ferramenta é especificada com orientação a objetos, usando *Unified Modeling Language* (UML). A linguagem é especificada utilizando definições regulares e a notação *Backus Naur Form* (BNF), sendo que a geração dos analisadores léxico e sintático é feita pelo Gerador de Analisadores Léxicos e Sintáticos (GALS). O resultado da compilação é um código intermediário que pode ser executado passo a passo. A ferramenta é desenvolvida observando os critérios de qualidade de software educacional definidos por Campos (1994) e utilizada durante o 1º semestre de 2005 pelos alunos da disciplina de Introdução à Programação do curso de Ciências da Computação da Universidade Regional de Blumenau (FURB).

Palavras-chave: Algoritmos. Compilador. Introdução à programação. Ensino-aprendizagem. Linguagens de programação.

ABSTRACT

The present assignment has by purpose describe the development of a tool that supports the teaching of introductory programming, allowing the development of algorithms in a structured programming language in Portuguese. The tool is specified with object orientation, using Unified Modeling Language (UML). The language was specified using regular definitions and the Backus Naur Form (BNF) notation. The lexical and syntactic analyzers generation is made by the Gerador de Analisadores Léxicos e Sintáticos (GALS). The compiling result is an intermediate code that can be executed step by step. The tool is developed observing the educational software quality criteria defined by Campos (1994) and it was used during the 1st semester of 2005 by the introduction to programming discipline students of the computer science course of Universidade Regional de Blumenau (FURB).

Keywords: Algorithms. Compiler. Introduction to programming. Education-learning. Programming Languages.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

O ensino de programação é essencial na grade curricular de um curso de ciência da computação, sendo considerado a base para o entendimento computacional. O ensino de programação acontece em várias disciplinas, específicas ou não sobre o assunto, sendo a primeira delas Introdução à Programação (ou nome similar), em geral oferecida no 1º semestre. Essa disciplina normalmente aborda os princípios da lógica de programação, objetivando que o aluno desenvolva a capacidade de análise e resolução de problemas descrevendo-os através de algoritmos. Segundo as diretrizes curriculares do MEC (MINISTÉRIO DA EDUCAÇÃO, 1999, p.6), "O desenvolvimento de algoritmos, juntamente com o estudo de estruturas de dados deve receber especial atenção na abordagem do tema programação."

Este processo de ensino apresenta dois grandes desafios. O primeiro é despertar a criatividade necessária para o desenvolvimento de soluções computacionais para os problemas. O segundo é representar a solução usando lógica de programação.

A forma usada para representar um algoritmo é variada, podendo destacar-se fluxograma e Portugol. Conforme Saliba (1994, p. 3), o fluxograma faz uso de símbolos geométricos que representam as estruturas de um programa. Estes símbolos são conectados por arestas dirigidas que fornecem a seqüência de execução. Já o Portugol[1] é uma linguagem que permite representar um algoritmo fazendo uso da língua portuguesa. Através de estruturas básicas (seqüência, seleção ou repetição), é possível construir programas usando uma sintaxe que se aproxima das linguagens de programação usuais. No entanto, independente da forma escolhida, em geral, a descrição do algoritmo é feita no papel.

Aliados à dificuldade para representar o algoritmo e ao uso do papel, surgem

[1] Guimarães e Lages (1994, p.19) explicam que o nome vem da junção de Português com Algol (Portu+gol).

questionamentos do tipo: Por que a solução proposta não é adequada? Qual o “caminho” que a solução proposta está seguindo? Estes questionamentos estão ligados ao fato de que a lógica de programação apresenta um grau de abstração inicial grande, pois muitas vezes o aluno não consegue visualizar o que aconteceria se a solução por ele proposta fosse executada em um computador.

Tendo em vista o problema apresentado, este trabalho descreve a implementação de uma ferramenta, utilizando a representação Portugol, para dar apoio ao ensino de introdução à programação, permitindo o desenvolvimento de algoritmos em uma linguagem de programação estruturada e em português. Na construção da ferramenta são utilizados padrões de qualidade para software educacional apresentados por Campos (1994, p. 121-122), que sugere a necessidade de mensagens de erro adequadas, de acesso a instruções sobre o uso da ferramenta (ajuda), de adequação ao currículo do curso, entre outras recomendações.

1.1 OBJETIVOS DO TRABALHO

O objetivo principal deste trabalho é desenvolver uma ferramenta para auxiliar no ensino de Introdução à Programação, disciplina do curso de Ciências da Computação da FURB.

Os objetivos específicos do trabalho são:

a) disponibilizar analisadores léxico, sintático e semântico para verificar os algoritmos elaborados;
b) efetuar detecção/tratamento de erros, emitindo mensagens capazes de auxiliar a correção dos algoritmos elaborados;
c) possibilitar a execução dos algoritmos passo a passo, com opção para visualizar os valores das variáveis declaradas;

d) utilizar os padrões de qualidade para o desenvolvimento de softwares educacionais recomendados por Campos (1994).

1.2 ESTRUTURA DO TRABALHO

O trabalho está organizado em quatro capítulos. No capítulo seguinte é descrita a fundamentação teórica utilizada para embasar este trabalho. É apresentada a importância do computador no ensino e suas classificações, os modelos (ou paradigmas) de linguagens de programação, a definição de compiladores, e os critérios que devem ser observados para se obter qualidade em software educacional. O capítulo é finalizado com os trabalhos correlatos.

O capítulo 3 traz a especificação e implementação da ferramenta. É feita uma avaliação da ferramenta considerando os critérios de qualidade de software educacional. Ao final do capítulo são apresentados os resultados alcançados a partir dos testes realizados pelos alunos e pelo professor da disciplina de Introdução à Programação.

O capítulo 4 contém a conclusão do trabalho, juntamente com sugestões para trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

Neste capítulo são apresentados alguns aspectos teóricos relacionados ao trabalho, tais como: ensino no computador, linguagens voltadas ao ensino de programação, linguagens de programação, compiladores, qualidade de software educacional. Na última seção são apresentados alguns trabalhos correlatos.

2.1 ENSINO NO COMPUTADOR

Segundo Galvis-Panqueva (1997), com a massificação da informática em empresas, lares e escolas, proliferaram os softwares voltados para a educação. Usa-se o computador na educação visando auxiliar o processo de ensino ou desenvolver melhores meios de aprendizado.

Baranauskas et al. (1999, p. 45-46) define três classes de sistemas computacionais em educação: ensino assistido por computador, aprendizado socialmente distribuído e ambientes interativos de aprendizagem.

2.1.1 Ensino assistido por computador

No ensino assistido por computador, o computador é visto como uma ferramenta para armazenamento, representação e transmissão de informação. Essas informações são divididas em módulos, que mostram o assunto de maneira gradual e seqüencial. E, geralmente após a apresentação de um módulo, o usuário é submetido a perguntas que devem ser respondidas de acordo com o material apresentado.

Este tipo de sistema surgiu na década de 60, porém, devido a tecnologia pouco

desenvolvida, era rígido e não despertava o interesse do aluno, sendo apenas uma nova forma de apresentar conteúdo, um computador ao invés de um livro. Na década de 70, foi desenvolvido o *Intelligent Computer Assisted Learning* (ICAI), no qual a seqüência da apresentação passa a ser personalizada de acordo com o conhecimento demonstrado pelo usuário, tornando o sistema mais atrativo. Estes sistemas foram evoluindo à medida que novas tecnologias e técnicas de inteligência artificial foram surgindo e hoje são chamados de *Intelligent Tutoring Systems* (ITS) ou Tutores Inteligentes (TI) (BARANAUSKAS et al., 1999, p. 47).

Baranauskas et al. (1999, p. 48) cita como exemplo de software do tipo ensino assistido por computador o Microsoft Windows 95, O Início, um tutorial do tipo passo a passo que dispõe de um módulo para iniciantes e um módulo para usuários avançados, nos quais pode-se aprender a utilizar o Windows 95. Neste tipo de software, o usuário tem a liberdade de definir o que deseja aprender, quando aprender e em que velocidade isto deve acontecer.

2.1.2 Aprendizado socialmente distribuído

O aprendizado socialmente distribuído surge com a Internet e a globalização da informação. Este tipo de sistema permite que o conhecimento possa estar numa área comum onde todos possam buscá-lo e ao mesmo tempo acrescentar novos conhecimentos. O potencial deste tipo de sistema é que ele é útil não só ao estudante, mas também ajuda na formação dos professores. A Internet é o meio utilizado para ligar o conhecimento de diversas pessoas e estes conteúdos estão disponível na *World Wide Web* (WWW).

2.1.3 Ambientes interativos de aprendizagem

Nos ambientes interativos de aprendizado, “o aprendizado é entendido como a construção individual do conhecimento a partir de atividades de exploração, investigação e descoberta.” (BARANAUSKAS et al., 1999, p. 50). Os ambientes interativos de aprendizado são baseados em quatro princípios: o estudante deve construir seu conhecimento; o controle do sistema é feito, de forma mais significativa, pelo estudante; o sistema é individualizado para cada estudante; e o *feedback* é gerado em função da interação do estudante com o ambiente. Algumas categorias desta classe de sistemas são: sistemas de modelagem e simulação, sistemas de autoria e ambientes e linguagens de programação.

Em um ambiente de modelagem e simulação típico, o usuário constrói um modelo de um fenômeno/objeto que deseja estudar, utilizando primitivas específicas para representação do modelo, presentes no ambiente computacional. Feito o modelo, o sistema o executa (simula) e apresenta os resultados da simulação. O usuário observa a simulação e pode analisar os resultados obtidos, comparando o modelo construído com o sistema real. Um exemplo deste tipo de sistema é o Constructor (TERZIAN, 2005), um jogo de simulação em que o usuário possui uma construtora e tem por objetivo construir uma cidade, dentro dos padrões definidos no início do jogo, atendendo às necessidades dos moradores. A tarefa é dificultada pelo fato de que existem outras construtoras que não pretendem permitir que o jogador construa sua cidade.

> Estamos chamando de “sistemas de autoria”, os sistemas computacionais para autoria de hipertexto; isto é, sistemas que permitem ao usuário não apenas ser o “leitor” de um documento, mas também ser um “escritor”, produzindo documentos de hipertexto. (BARANAUSKAS et al., 1999, p. 57).

Um exemplo apresentado por Baranauskas et al (1999, p.58) é o Hyperstudio, um sistema utilizado para criar hipertextos com objetos tais como textos, sons, figuras e imagens animadas, que são interligados por meio de conexões. O usuário pode alterar entre o modo

escritor, quando elabora o documento, e leitor, quando habilita a opção para visualizar seu hipertexto.

Por fim, os ambientes de programação possuem um grande destaque como ferramenta educacional, uma vez que proporcionam a oportunidade de não só verificar o aprendizado do aluno, mas também de observar como o aluno chegou a determinado resultado. Neles o aluno descreve a solução de um problema, utilizando uma linguagem de programação. Uma das linguagens de programação mais vastamente utilizada com objetivos educacionais é a linguagem Logo. Nesta linguagem o aluno faz desenhos geométricos, através de comandos que determinam as ações de uma tartaruga (VALENTE; VALENTE, 1988).

2.2 LINGUAGENS VOLTADAS AO ENSINO DE PROGRAMAÇÃO

Escolher a linguagem mais adequada para o ensino de programação é uma tarefa que requer muito cuidado e análise. Baranauskas et al (1999, p.54) destaca o Prolog, Logo e Pascal. Estas linguagens possuem a característica de serem utilizadas para o ensino.

Prolog (*PROgramming in LOGic*) é uma linguagem de programação lógica, declarativa e não-procedural. Palazzo (1997, p. 20) afirma que a primeira implementação da linguagem foi realizada por Alin Colmerauer e sua equipe, na Universidade de Aix-Marseille em 1972. A idéia do Prolog é que o programador ao invés de dizer ao computador o que deve ser feito, ele descreve o objeto que deve ser computado. Em outras palavras, a tarefa do programador passa a ser simplesmente a especificação do problema que deve ser solucionado. O objetivo deste tipo de linguagem é estimular o raciocínio lógico, abstraindo o formalismo de linguagens de programação. Um programa em Prolog é feito sobre um conjunto de cláusulas, cada uma sendo um fato ou uma regra. Um fato denota uma verdade incondicional, enquanto que as regras definem as condições que devem ser satisfeitas para que certa

declaração seja considerada verdadeira.

A linguagem de programação Logo (VALENTE; VALENTE, 1988) caracteriza-se por ser uma tentativa de tornar o conceito de programar algo de fácil entendimento. Valente (1996, p. 4-5) afirma que o Logo foi desenvolvido na França em 1967, ficando porém restrito a laboratórios de pesquisa até 1976. Neste ano, o professor Papert do *Massachusetts Institute of Technology* (MIT) realizou um projeto que introduziu o uso do Logo nas escolas. Logo faz uso da imagem de uma tartaruga capaz de caminhar e deixar rastros. Trabalhando com comandos que permitem determinar a posição e direção da tartaruga e fazendo uso de uma sintaxe simples, possibilita uma fácil assimilação sobre o uso da ferramenta por parte do usuário. Possui a representação gráfica do rastro da tartaruga e um editor que mostrar os comandos gerados a partir do rastro.

Pascal (FARRER et al., 1999), uma versão aprimorada do Algol-60, foi projetada por Niklaus Wirth para o ensino. Tornou-se popular em meados da década de 70, através de seu uso por parte das universidades para o ensino de programação. Programar em Pascal significa utiliza-se de um conjunto de recursos (repetição, seleção, atribuição), juntamente com algumas regras de sintaxe, para descrever a solução de problemas. Para Sebesta (2000), a popularidade do Pascal está baseada na combinação de simplicidade e expressividade.

2.3 LINGUAGENS DE PROGRAMAÇÃO

O meio mais eficaz de comunicação entre as pessoas é a linguagem (língua ou idioma). Uma linguagem de programação serve como meio comunicação entre o indivíduo que deseja resolver um problema e o computador que irá ajudá-lo (PRICE; TOSCANI, 2001, p.1).

Sebesta (2000, p.18-20) afirma que o estudo de linguagens de programação traz uma lista de benefícios, tais como: aumento da capacidade de se expressar; maior conhecimento

para a escolha da linguagem mais apropriada para uma aplicação; aumento da capacidade para aprender novas linguagens; maior entendimento da importância da implementação e melhora na capacidade de projetar novas linguagens.

As linguagens de programação são classificadas por Lisbôa (2004), prioritariamente, em dois modelos:

a) modelo declarativo: as linguagens não possuem comandos, apenas “roteiros” que definem o que deve ser computado, de forma independente das manipulações que devem ser feitas para a obtenção dos resultados;
b) modelo imperativo: as linguagens expressam seqüências de comandos ou ações que, quando executados, realizam transformações sobre os dados armazenados na memória, para a obtenção de um resultado. Os recursos centrais das linguagens imperativas são as variáveis, as instruções de atribuição e repetição.

Com base nestes modelos foram criadas subclasses, que são conhecidas como paradigmas de linguagens de programação.

O modelo declarativo é dividido em: funcional, baseado na chamada recursiva de funções, sendo que o exemplo mais difundido é a linguagem LISP; e lógico, onde a computação é baseada na lógica matemática, tal como o Prolog.

O modelo imperativo é baseado na arquitetura Von Neumann e tem como subclasses:

a) linguagens orientadas a procedimento, onde as funcionalidades do programa são encapsuladas em módulos ou procedimentos, que podem ser executados de forma seqüencial ou concorrente. Caso a linguagem seja seqüencial, pode ser classificada em estruturada e não estruturada. Pode-se citar como exemplo Pascal, Fortran e C;
b) linguagens orientadas a objetos, onde a computação é vista como uma interação entre objetos, que se comunicam através da troca de mensagens. Exemplifica este tipo o Java.

2.4 COMPILADORES

"Um compilador é um programa que lê um programa escrito numa linguagem – a linguagem fonte – e o traduz num programa equivalente numa outra linguagem – a linguagem alvo" (AHO; SETHI; ULLMAN, 1995, p. 1). Aho, Sethi e Ullman (1995, p. 1) afirmam que os compiladores surgiram no início dos anos 50. A data precisa é difícil fornecer, pois existiam várias pesquisas sendo realizadas neste período e várias implementações foram feitas por diversos grupos.

De forma geral, a compilação é divida em dois processos: a análise e a síntese. A parte da análise consiste em dividir o código fonte em partes e verificar suas propriedades. A síntese constrói o programa alvo. O processo de análise engloba as análises léxica, sintática e semântica; enquanto que o processo de síntese engloba o gerador de código intermediário, o otimizador de código e o gerador de código objeto. A união das partes dos processos de análise e síntese formam o que é chamado de compilador conceitual, onde os processo são executados de forma seqüencial.

2.4.1 Análise léxica

O primeiro passo a ser executado é a análise léxica, que fragmenta o código fonte em *tokens* (símbolos básicos da linguagem), classificando-os em categorias. Após reconhecido, cada *token* pode ser classificado como palavra reservada, identificador, constante ou símbolo especial, entre outros. Nesta etapa, são ignorados os espaços em branco e os comentários.

Caso seja encontrado um *token* que não esteja de acordo com a especificação da linguagem, deve ser informado ao usuário que foi detectado um erro léxico no programa fonte.

2.4.1.1 Especificação dos *tokens*

A especificação de *tokens* pode ser feita através da definição de expressões regulares. Jargas (2001) define expressão regular como “uma composição de símbolos, caracteres com funções especiais, que, agrupados entre si e com caracteres literais, formam uma seqüência, uma expressão”. Quando se deseja atribuir um nome a uma expressão regular, para referir-se a ela, a expressão é chamada definição regular.

A especificação dos *tokens* consiste então em efetuar a definição regular do conjunto de caracteres aceitos pela linguagem. Para esta definição são utilizados operadores unários ou binários: o operador * indica repetição ou concatenação sucessiva de caracteres zero ou mais vezes; + indica concatenação sucessiva de caracteres uma ou mais vezes; o operador | indica escolha entre caracteres e o uso de caracteres entre colchetes indica a ocorrência de um elemento do conjunto especificado, podendo existir caracteres únicos ou faixas, considerando a ordem dos valores na tabela ASCII. Por exemplo, [0-9] especifica o conjunto composto pelos caracteres 0, 1, 2, 3, 4, 5, 6, 7, 8, 9.

O quadro 1 apresenta uma especificação de um identificador de uma linguagem.

```
letra : [a-z]
dígito : [0-9]
identificador : letra  (letra | dígito)*
```

Quadro 1 – Exemplo de definições regulares

Neste exemplo, uma letra é qualquer caractere entre a e z, dígito é um caractere entre 0 e 9 e o identificador começa com uma letra que pode ser seguida por zero ou mais letras ou dígitos.

2.4.2 Análise sintática

O processo seguinte à análise léxica é a análise sintática. A análise sintática tem por

função verificar a seqüência dos *tokens* recebida da análise léxica e montar uma árvore de derivação com base na gramática especificada para a linguagem.

A árvore de derivação pode ser construída utilizando-se duas técnicas. A primeira técnica é a estratégia *top-down* (descendente), onde a árvore é construída a partir do símbolo inicial da gramática, fazendo a árvore crescer até atingir as folhas. A segunda estratégia é *bottom-up* (redutiva) que faz o contrário, inicia a partir das folhas buscando encontrar ao final a raiz da árvore (PRICE; TOSCANI, 2001, p. 29).

Na análise sintática, um dos pontos importantes é a detecção de erros, em virtude de que uma boa parte da detecção e recuperação de erros de um compilador gira em torno desta etapa (AHO; SETHI; ULLMAN, 1995, p. 73). Um erro sintático ocorre quando a seqüência de *tokens* não está na ordem definida na gramática da linguagem. Caso isto ocorra, deve ser informado ao usuário que foi encontrado um erro sintático.

2.4.2.1 Especificação das regras sintáticas

Para a especificação da gramática da linguagem pode ser usada a notação *Backus-Naur Form* (BNF). Sebesta (2000, p. 116) diz que a notação BNF foi desenvolvida por Jonh Backus e posteriormente ligeiramente modificada por Peter Naur, por isso o nome BNF. Seu desenvolvimento foi feito para especificar a linguagem Algol 58.

Usa-se a BNF para descrever a sintaxe de uma linguagem, fazendo uso de abstrações. Um comando de saída de dados pode ser, por exemplo, representado pela abstração <escrita>, apresentada no quadro 2 (o sinal de maior e menor é utilizado para delimitar a abstração).

```
<escrita> → escreva ( <item> )
<item> → identificador | numero
```

Fonte: adaptado de Sebesta (2000, p. 116)

Quadro 2 – Exemplo de regras de produção

O símbolo à esquerda da seta é a abstração, e os símbolos após a seta são a definição

da abstração. Observa-se que esta definição, chamada de regra ou produção, pode conter outras abstrações ou *tokens*.

Segundo Sebesta (2000, p. 116), a notação BNF, apesar da simplicidade pode ser utilizada para descrever a maioria das linguagens de programação.

2.4.3 Análise semântica

A análise semântica avalia a árvore criada pela análise sintática, buscando determinar o comportamento que o programa terá durante a execução. Louden (2004, p. 259) afirma que a análise semântica pode ser dividida em duas categorias. A primeira é a análise de um programa com base nas regras da linguagem, a fim de verificar sua correção e garantir sua execução. A segunda categoria de análise semântica é aquela efetuada com o objetivo de melhorar a eficiência da execução do programa, isto é, fazer uma otimização do código.

Além disso, segundo Louden (2004, p. 10), a semântica de um programa pode ser estática ou dinâmica. Em sua maioria, as linguagens têm atributos que podem ser determinados antes da execução, durante a análise semântica. Isto caracteriza a semântica estática de um programa. A semântica estática é responsável pela verificação da coerência da declaração e do uso de identificadores, entre outras atividades. Já a semântica dinâmica, que irá acontecer somente durante a execução do programa, faz, por exemplo, a verificação do tipo de um dado digitado pelo usuário.

O analisador semântico também tem por função detectar e tratar erros semânticos.

2.4.3.1 Esquemas de tradução

Price e Toscani (2001, p. 86) afirmam que esquema de tradução “é uma extensão de

uma gramática livre do contexto, extensão esta realizada através da associação de atributos aos símbolos gramaticais e de ações semânticas às regras de produção.” Um atributo pode conter qualquer tipo de informação e seu valor é definido por uma ação semântica. Ações semânticas associadas a uma regra de produção da gramática são executadas quando a regra é analisada, considerando que as ações semânticas podem produzir efeitos colaterais tais como imprimir um valor, gerar código, armazenar um literal em uma tabela ou em arquivo, emitir uma mensagem de erro, entre outros.

O quadro 3 apresenta um esquema de tradução com as ações semânticas especificadas à direita da regra de produção. Com isto, cada ação será executada após o reconhecimento da produção a qual está associada.

SINTAXE	**EXEMPLO**
<declaração> → identificador : <tipo> **#1** <tipo> → real **#2** | integer **#3**	média : real
ação#1 : TS.adiciona (identificador.lexema, <tipo>.valor) **ação#2 :** <tipo>.valor:= “r” **ação#3 :** <tipo>.valor:= “i”	

Fonte: adaptado de Price e Toscani (2001, p. 85)

Quadro 3 – Exemplo de ações semânticas

Quando a segunda ou a terceira produção for analisada, o atributo valor associado ao não-terminal <tipo> recebe o tipo reconhecido (r ou i). E após a análise da primeira produção, a ação **#1** vai adicionar o identificador juntamente com seu tipo na tabela de símbolos.

É possível observar a ordem de execução das ações semânticas, desenhando-se a árvore de derivação e incluindo nela as ações semânticas. Considerando o esquema de tradução apresentado no quadro 3 e o exemplo de declaração de variável, tem-se a árvore de derivação apresentada na figura 1.

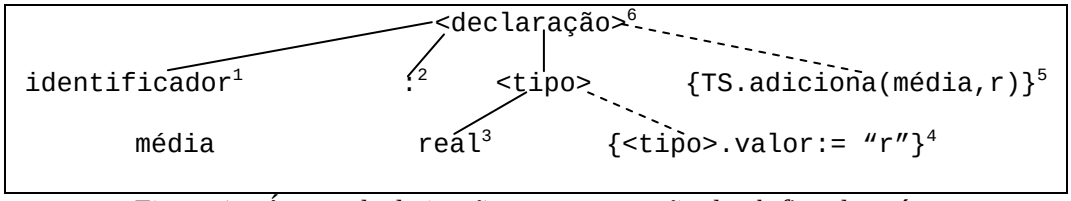


Figura 1 – Árvore de derivação com numeração *depth-first* dos nós

2.4.4 Geração de código intermediário

A geração de código intermediário pode ser feita juntamente com a análise semântica. Price e Toscani (2001, p. 115) definem a geração de código intermediário como “a transformação da árvore de derivação em um segmento de código”, que pode ser interpretado por uma máquina. Esta etapa apresenta algumas vantagens descritas por Price e Toscani (2001, p. 115): possibilita a obtenção de um código objeto final mais eficiente; simplifica a implementação do compilador; e possibilita a tradução do código intermediário para diversas máquinas.

Esta etapa é implementada em virtude da complexidade da geração de código de máquina e também pelo fato de que alguns compiladores não geram código executável. Em vez disso, geram código intermediário que é interpretado por um outro programa.

O código intermediário pode ter várias representações, entre elas a notação polonesa. José Neto (1987, p. 141) explica que esta notação utiliza uma pilha “onde os operandos são armazenados até que um operador force sua manipulação, operação e desempilhamento”. A notação polonesa apresenta duas variações: a pré-fixada e a pós-fixada. Na notação pré-fixada o operador aparecer antes dos operandos. Já na notação pós-fixada os operadores aparecem depois dos operandos, que são empilhados até que seja encontrado um operador n-ário. Neste momento é efetuada a operação utilizando os n operandos do topo da pilha. A expressão `(a+b)*c`, por exemplo, é representada como `ab+c*` na notação pós-fixada. O quadro 4 apresenta a execução dessa expressão considerando que os valores de a, b, c são respectivamente 2, 3 e 4.

EXECUÇÃO	PILHA	
ab+c*		
a	2	
ab	2	3
ab+	5	
ab+c	5	4
ab+c*	20	

Quadro 4 – Exemplo notação pós-fixada

2.5 QUALIDADE DE SOFTWARE EDUCACIONAL

Pressman (2002, p. 193) define qualidade de software como “o conjunto das adequações dos requisitos funcionais e de desempenho, documentação clara dos padrões de desenvolvimento e características implícitas esperadas por todos os profissionais de desenvolvimento de software”. Porém, a importância dos itens deste conjunto pode variar de acordo com o tipo da aplicação e do usuário, devendo-se destacar que são necessários métodos para a avaliação desta qualidade.

Diversos autores como Boehm e Schmauch, todos citados por Pressman (2002, p. 191-194), discutem os aspectos para a composição da avaliação da qualidade de software, sendo que cada um deles sugere um modelo. Quando da definição de um método, deve-se observar cada um dos modelos e escolher o que melhor se adequada ou fazer uso de trechos de vários modelos. O método desenvolvido por Campos (1994) utiliza-se de características dos modelos de Boehm e McCall, porém de forma mais abrangente e descendo a níveis de refinamento mais baixos.

Campos (1994, p. 96-97) sugere um modelo que determina a qualidade de um software educacional. No entanto, estabelecer a qualidade em um software educacional é uma tarefa complexa que envolve inúmeros e diversificados atributos. A avaliação deve ser efetuada tanto na fase de concepção (projeto e produção), quanto na fase de utilização (uso educacional) (CAMPOS, 1994, p. 93). De forma geral, os critérios que devem ser observados são:

a) o acesso a instruções sobre o uso (ajuda, *help*) deve ser independente da situação em que o usuário se encontra;
b) as mensagens de erro devem ser adequadas, facilitando a identificação de problemas;

c) o *feedback* deve ser fornecido em todas as situações, informando o que está ocorrendo no momento;

d) o software tem que ser adequado ao currículo do curso, ou seja, não devem ser necessárias mudanças no currículo para a utilização do software. Também deve haver uma preocupação de adaptabilidade ao nível do usuário e uma integração do software com outros recursos instrucionais;

e) os resultados finais devem ser claros, permitindo que o usuário saiba exatamente qual o resultado final de suas ações;

f) caso ocorra um erro, deve ser permitida sua correção, sem eliminar o que foi feito;

g) a leitura da tela deve ser fácil, isto é, não deve ser necessário um esforço por parte do usuário para entender o que deve ser feito naquele momento;

h) a ferramenta deve despertar o interesse do aluno;

i) ilustrações e cores devem ser usadas para facilitar a percepção do usuário e destacar informações importantes.

2.6 TRABALHOS CORRELATOS

Várias foram as ferramentas desenvolvidas visando auxiliar no processo de aprendizado da lógica de programação, entre elas destacam-se: o Ambiente de Apoio ao Aprendizado de Programação (AMBAP), o Ambiente de Simulação e Animação de Algoritmos (ASA), o software para o auxílio ao aprendizado de algoritmos e a ferramenta de apoio ao ensino de algoritmos (CIFluxProg).

O AMBAP, descrito por Almeida (2002), surgiu a partir de um projeto de pesquisa desenvolvido na Universidade Federal de Alagoas (UFAL). Foi concebido e desenvolvido para auxiliar o aluno iniciante no aprendizado de programação. O ambiente permite ao aluno

construir seu programa numa linguagem algorítmica, executando-o e tendo a oportunidade de entender os conceitos relacionados à construção de algoritmos, através de um processo de simulação. Possui uma liberdade sintática e até léxica, considerando por exemplo que “faça” é igual a “faca” e que a estrutura enquanto-faça pode ser escrita como faça-enquanto.

O ASA, criado a partir de pesquisa realizada pelo SENAC, utiliza a representação de algoritmos na forma de fluxogramas, com a opção de mapear os algoritmos em diversas representações (fluxogramas, planos, Pascal, C, Clipper e pseudocódigo). Os modos de execução podem ser: passo a passo, contínuo ou com pontos de parada (USP, 2004).

Tagliari (1996) descreve o protótipo de um software que ajuda o aluno no estudo de algoritmos através de exemplos pré-definidos. No software para o auxílio ao aprendizado de algoritmos o usuário pode executar os algoritmos pré-definidos, visualizando sua estrutura e seu funcionamento, bem como, o conteúdo das variáveis. Não é permitido ao aluno escrever seu próprio código.

O CIFluxProg, que significa construtor e interpretador de algoritmos para programação, foi desenvolvida na Universidade do Vale do Itajaí (UNIVALI). Segundo Santiago e Dazzi (2004), é composto por dois ambientes gráficos: um para desenvolvimento de algoritmos utilizando fluxogramas e um outro para descrição de algoritmos na forma textual. Permite que, durante a execução, possam ser visualizados os valores de cada variável do algoritmo. A sintaxe utilizada na ferramenta é parecida com a sintaxe da linguagem C.

3 DESENVOLVIMENTO DO TRABALHO

Neste capítulo são descritos os requisitos, a especificação da ferramenta e da linguagem. Também é apresentada a implementação detalhando a operacionalidade da ferramenta, a avaliação da qualidade e os testes realizados pelos alunos e pelo professor da disciplina de Introdução à Programação. Finaliza com uma descrição dos resultados obtidos.

3.1 REQUISITOS PRINCIPAIS DO PROBLEMA A SER TRABALHADO

Uma ferramenta para auxiliar no ensino de Introdução à Programação deve possuir um editor para digitação de algoritmos, com as características de um editor de texto convencional, tendo comandos para edição de textos (recortar, copiar, colar, selecionar tudo), manipulação de arquivos (abrir, salvar, salvar como), entre outros. Em sua interface e em seu modo de interação com o usuário devem ser seguidas as recomendações do padrão de qualidade de software educacional sugerido por Campos (1994).

A ferramenta deve compilar os algoritmos escritos em uma linguagem de programação estruturada e em português, efetuando as análises léxica, sintática e semântica dos algoritmos digitados e informando os possíveis erros detectados. Caso não seja encontrado nenhum erro, deve ser gerado um código intermediário.

Depois de compilado, um algoritmo pode ser executado[2] até o fim ou passo a passo, sendo que a execução será feita com base no código intermediário gerado. No decorrer da execução passo a passo deve ser possível acompanhar o valor de cada variável declarada no algoritmo.

[2] Os algoritmos serão executados por um interpretador. Segundo Price e Toscani (1995, p.5), interpretadores "são processadores que aceitam como entrada o código intermediário de um programa anteriormente traduzido e produzem o efeito de execução do algoritmo original".

3.2 ESPECIFICAÇÃO

Os tópicos seguintes descrevem a especificação da ferramenta. Inicialmente é apresentada a especificação da linguagem de programação estruturada e em português e a especificação da linguagem intermediária. Em seguida, são apresentadas as funcionalidades da ferramenta na forma de casos de uso juntamente com diagramas de atividades. Também são descritas as principais classes modeladas.

3.2.1 Especificação da linguagem fonte

A linguagem especificada é seqüencial e imperativa. Foi definida de acordo com a sintaxe utilizada pelo professor Paulo Dias na disciplina de Introdução à Programação do curso de Ciências da Computação da FURB. Possui as seguintes características:

a) comandos em língua portuguesa;
b) estrutura semelhante ao Pascal com declaração de variáveis, declaração de tipos (somente do tipo matriz), comando de entrada via teclado, comando de saída em vídeo, comando de atribuição, comandos de seleção (se-então, escolha-caso) e comandos de repetição (enquanto-faça, repita-até, para-faça);
c) a linguagem é *case sensite*, porém as palavras reservadas não são *case sensite*.

A linguagem foi especificada usando a ferramenta GALS (GESSER, 2003).

3.2.1.1 Especificação dos *tokens*

A primeira etapa na definição da linguagem consiste na especificação das definições regulares, que servem como expressões auxiliares para a definição dos *tokens*. O quadro 5

apresenta as definições regulares da linguagem[3].

```
LETRA : [A-Za-zÓóéÉçÇãÃíÍáÁúÚâÂêÊîÎôÔûÛõÕàÀüÜ]
DIGITO  : [0-9]
COMENTARIO : "{" [^"}"]*"}"
```

Quadro 5 – Definições regulares

Em seguida é feita a definição dos *tokens*. Nesta etapa devem ser definidos os identificadores, as constantes (numéricas e alfanuméricas), as palavras reservadas, os símbolos especiais e os comentários.

Os identificadores da linguagem iniciam com uma letra e podem conter uma seqüência de letras, dígitos ou o caractere *underline* (_), sendo que nos identificadores as letras maiúsculas e minúsculas são diferenciadas (*case sensitive*). Esta definição é mostrada no quadro 6.

```
IDENTIFICADOR : {LETRA}({LETRA}|{DIGITO}|"_" )*
```

Quadro 6 – Identificadores

As palavras reservadas foram definidas como casos especiais de identificadores, como pode ser visto no quadro 7. A relação completa das palavras reservadas encontra-se no apêndice A.

```
algoritmo = IDENTIFICADOR : "algoritmo"
inicio = IDENTIFICADOR : "inicio"
fim = IDENTIFICADOR : "fim"
tipo = IDENTIFICADOR : "tipo"
matriz = IDENTIFICADOR : "matriz"
```

Quadro 7 – Palavras reservadas

A linguagem possui constantes numéricas inteiras e reais: as constantes inteiras possuem apenas dígitos e as constantes reais são compostas por dígito(s), o caractere ponto (**.**) e outro(s) dígito(s). Possui também constantes do tipo cadeia, que são delimitadas por **‘**. Constantes deste tipo podem conter qualquer caractere, com exceção de quebra de linha. No quadro 8 encontra-se a especificação das constantes da linguagem.

[3] O uso de aspas duplas indica a ocorrência literal da seqüência de caracteres especificada. Por exemplo, “{” indica a ocorrência de abre chaves. O uso de [^] indica qualquer caractere exceto os caracteres dentro dos colchetes, após o símbolo ^.

```
CONSTANTE_INTEIRO : {DIGITO}+
CONSTANTE_REAL : {DIGITO}+ "." {DIGITO}+
CONSTANTE_LITERAL : ' ([^'\n])* '
```

Quadro 8 – Constantes

Os operadores aritméticos e relacionais são definidos como símbolos especiais[4] e são apresentados no quadro 9.

operador	**descrição**
=	igual
<>	diferente
<	menor que
>	maior que
<=	menor que ou igual a
>=	maior que ou igual a
+	adição
-	subtração
*	multiplicação
/	divisão

Quadro 9 – Operadores aritméticos e relacionais

A linguagem possui apenas comentários de bloco, que são quaisquer seqüências de caracteres que estiverem entre chaves. Comentários de bloco bem como caracteres de formatação[5] devem ser ignorados pelo analisador léxico, tal como especificado no quadro 10.

```
: [" "\t\n\r]*
: {COMENTARIO}
```

Quadro 10 – Caracteres de formatação e comentários de bloco

3.2.1.2 Especificação sintática

O próximo passo consiste na especificação da gramática para definir a sintaxe da linguagem. A linguagem é estruturada e não possui suporte a procedimentos, portanto, possui somente um bloco, que inicia com a palavra **algoritmo** e termina com **fim**. O quadro 11 apresenta a sintaxe da linguagem.

[4] Outros símbolos especiais da linguagem são: **. ; , ( ) : := [ ] ..**

[5] São caracteres de formatação: \n que significa nova linha; \t que significa tabulação horizontal; \r que significa retorno de cursor; “ ”que significa espaço em branco.

```
<algoritmo> → algoritmo IDENTIFICADOR ;
                    <declaracao_tipo>
                    <declaracao_variaveis>
              inicio <lista_comandos> fim .

<declaracao_tipo> → ε |
  tipo <lista_identificadores> = matriz [<faixa>] <tipo_primitivo> ; <declaracao_tipo>
<faixa> → CONSTANTE_INTEIRO .. CONSTANTE_INTEIRO

<declaracao_variaveis> → var <variaveis> | ε
<variaveis> →<lista_identificadores> : <tipo> ; |
              <lista_identificadores> : <tipo> ; <variaveis>
<tipo> → IDENTIFICADOR | <tipo_primitivo>
<tipo_primitivo> → inteiro | real | logico | caractere | cadeia
<lista_identificadores> → IDENTIFICADOR | IDENTIFICADOR , <lista_identificadores>

<lista_comandos> → <comando> | <comando> <lista_comandos>
<comando> →  <atribuicao> | <entrada> | <saida> | <repeticao> | <selecao> | <limparTela>

<atribuicao> → IDENTIFICADOR:= <expressao>; | IDENTIFICADOR [<expressao>]:= <expressao>;

<entrada> → leia ( <lista_entrada> );
<lista_entrada> → <variavel_entrada> | <variável_entrada> , <lista_entrada>
<variavel_entrada> → IDENTIFICADOR  | IDENTIFICADOR [<expressao>]

<saida> → escreva ( <lista_saida> );
<lista_saida> → <item> | <item> , <lista_saida>
<item> → <variavel_saida> | <numero> |  CONSTANTE_LITERAL
<variavel_saida> → IDENTIFICADOR | IDENTIFICADOR [<expressao>]

<repeticao> → <enquanto> | <repita> | <para>
<enquanto> → enquanto <expressao> faca <lista_comandos> fimenquanto;
<repita> → repita <lista_comandos> ate <expressao>;
<para>→ para IDENTIFICADOR de <expressao> ate <expressao> faca <lista_comandos> fimpara;

<selecao> → <se> | <escolha>
<se> → se <expressao> entao <lista_comandos> <senao> fimse ;
<senao> → senao <lista_comandos> | ε
<escolha> → escolha IDENTIFICADOR <casos> <senao> fimescolha ;
<opcao> → CONSTANTE_INTEIRO | CONSTANTE_LITERAL
<casos> → caso <opcao> : <lista_comandos> | caso <opcao> : <lista_comandos> <casos>

<limparTela> → limparTela ;

<expressao> → <expressao_aritmetica_logica> | <relacional>
<relacional> →  <expressao_aritmetica_logica> = <expressao_aritmetica_logica> |
                <expressao_aritmetica_logica> <> <expressao_aritmetica_logica>|
                <expressao_aritmetica_logica> < <expressao_aritmetica_logica> |
                <expressao_aritmetica_logica> > <expressao_aritmetica_logica> |
                <expressao_aritmetica_logica> <= <expressao_aritmetica_logica>|
                <expressao_aritmetica_logica> >= <expressao_aritmetica_logica>
<expressao_aritmetica_logica> → <termo> <menor_prioridade>
<menor_prioridade> → + <termo> <menor_prioridade> | - <termo><menor_prioridade> | ε
<termo> → <elemento> <maior_prioridade>
<maior_prioridade> → * <elemento> <maior_prioridade> | / <elemento> <maior_prioridade> |
              div <elemento> <maior_prioridade> | e <elemento> <maior_prioridade> |
              ou <elemento> <maior_prioridade> | mod <elemento> <maior_prioridade> | ε
<elemento> → <variavel> | <numero> | CONSTANTE_LITERAL | <constante_logica>  |
              (<expressao>) | <sinal> <elemento> | abs (<expressao>) | int (<expressao>)
<variavel> → IDENTIFICADOR  | IDENTIFICADOR [<expressao>]
<constante_logica> → verdadeiro |  falso
<numero> → CONSTANTE_INTEIRO | CONSTANTE_REAL
<sinal> → + | -
```

Quadro 11 – Sintaxe da linguagem

3.2.1.3 Especificação semântica

Juntamente com a sintaxe podem ser definidas as ações semânticas usando esquemas de tradução. Basta incluir na gramática o símbolo sustenido (#) acompanhado de um número que irá identificar a ação. Nos quadros 12 e 13 encontram-se os esquemas de tradução para o algoritmo e para o comando de saída de dados da linguagem bem como um exemplo. No apêndice B é apresentada a gramática com as demais ações semânticas.

SINTAXE	**EXEMPLO**
<algoritmo> → algoritmo IDENTIFICADOR **#1** ; <declaracao_tipo> <declaracao_variaveis> inicio **#2** <lista_comandos> fim . **#3**	**ALGORITMO** com_saída_de_dados ; **INÍCIO** **ESCREVA** ('oi mundo!'); **FIM** .
ação#1: Adicionar o nome do algoritmo na tabela de símbolos. **ação#2:** Informar a linha onde iniciam os comandos do algoritmo. **ação#3:** Gerar a instrução STP (final do algoritmo). Informar se alguma variável não foi utilizada ou não foi inicializada.	

Quadro 12 – Esquema de tradução do algoritmo

SINTAXE	**EXEMPLO**
<saida> → escreva (<lista_saida>) ; <lista_saida> → <item> **#21** | <item> **#21** , <lista_saida> <item> → <variavel_saida> | <numero> | CONSTANTE_LITERAL **#22** <variavel_saida> → IDENTIFICADOR **#24** | IDENTIFICADOR **#102** [<expressao> **#103**] **#23**	**ALGORITMO** com_saída_de_dados ; **INÍCIO** **ESCREVA** ('oi mundo!'); **FIM** .
ação#21: Gerar instrução WRT (escrever valor). **ação#22:** Gerar instrução LDS (carregar uma constante literal). **ação#23:** Gerar instrução LDA (carregar o valor de uma variável do tipo matriz). Alterar na tabela de símbolos a propriedade "usado" do identificador (reconhecido pela ação #102) para verdadeiro. **ação#24:** Verificar se o identificador foi declarado (existe na tabela de símbolos) e se é variável de tipo primitivo. Em caso afirmativo, gerar instrução LDV (carregar o valor da variável) e alterar na tabela de símbolos a propriedade "usado" do identificador para verdadeiro. **ação#102:** Verificar se o identificador foi declarado (existe na tabela de símbolos) e se é variável do tipo matriz. Se for, armazenar o identificador para uso em outra ação semântica. **ação#103:** Verificar se a expressão é do tipo inteiro. Se não for, informar incompatibilidade de tipos (o índice de uma matriz só pode ser do tipo inteiro).	

Quadro 13 – Esquema de tradução do comando de saída

3.2.1.4 Especificação da linguagem intermediária

A linguagem intermediária é composta por um conjunto de instruções, onde cada instrução tem quatro campos. O primeiro é o número da instrução, cujo valor é utilizado para referenciar a instrução que deve ser executada. O segundo campo é o código da instrução, que é formado por até quatro letras maiúsculas. O terceiro é o parâmetro da instrução. E o quarto indica em qual linha do algoritmo encontra-se o comando que corresponde à instrução, sendo que um comando pode corresponder a várias instruções na linguagem intermediária.

No total foram especificadas 41 instruções, sendo algumas delas apresentadas no quadro 14. A lista completa por ser vista no apêndice C.

CÓDIGO	PARÂMETRO	FUNCIONAMENTO
LDS	constante	Alocar uma posição na memória. Armazenar a constante passada por parâmetro nesta posição.
STP	0	Encerrar a execução.
WRT	0	Escrever na tela o valor armazenado na última posição da memória. Desalocar a última posição da memória.

Quadro 14 – Código intermediário

No quadro 15 é apresentado um exemplo de algoritmo e seu respectivo código intermediário.

ALGORITMO	CÓDIGO INTERMEDIÁRIO
ALGORITMO com_saída_de_dados ; **INÍCIO** **ESCREVA** ('oi mundo!'); **FIM** .	0 LDS 'oi mundo!' 3 1 WRT 0 3 2 STP 0 4

Quadro 15 – Exemplo de algoritmo e seu respectivo código intermediário

A execução deste código é feita da seguinte forma: o valor passado por parâmetro na instrução LDS (‘oi mundo!’) é carregado para o topo da memória, que é uma pilha; a instrução WRT escreve na tela o valor que está no topo da memória e desaloca o topo da memória; a instrução STP encerra a execução do algoritmo.

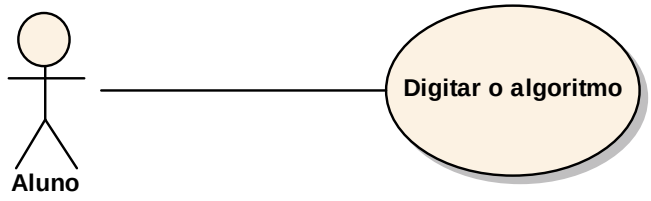
3.2.2 Especificação da ferramenta

A ferramenta foi especificada com orientação a objetos, usando *Unified Modeling Language* (UML). Foi utilizado o Enterprise Architect para o desenvolvimento dos diagramas de casos de uso (*use cases*), de atividades e de classes.

3.2.2.1 Casos de uso e diagramas de atividades

A ferramenta possui três casos de uso: digitar, compilar e executar o algoritmo.

A saída do processo de digitação do algoritmo serve como entrada para os processos seguintes. Para realizar este processo o usuário possui um editor para a digitação de um novo algoritmo com a opção de salvar o mesmo, além da opção para abrir um algoritmo já existente. O quadro 16 e a figura 2 apresentam o caso de uso e o diagrama de atividades para a digitação do algoritmo.



UC.1. Digitar o algoritmo: o aluno deve digitar o algoritmo, seguindo a sintaxe da linguagem (estruturado e em português) ensinada pelo professor da disciplina de Introdução à Programação.	
Pré-condições	Não possui.
Fluxo principal	1. Selecionar a opção **Novo.** 2. Digitar o algoritmo. 3. Selecionar a opção **Salvar**, informando um nome para o algoritmo.
Fluxo alternativo	1. Existe a opção de abrir um algoritmo já existente. Para isso, deve ser selecionada a opção **Abrir** e informado o nome do arquivo que contém o algoritmo desejado.
Fluxo de exceção	Não há.
Pós-condições	Será habilitada a opção **Compilar**.
Requisitos atendidos	1. Possuir um editor para digitação de algoritmos.

Quadro 16 - Caso de uso: digitar o algoritmo

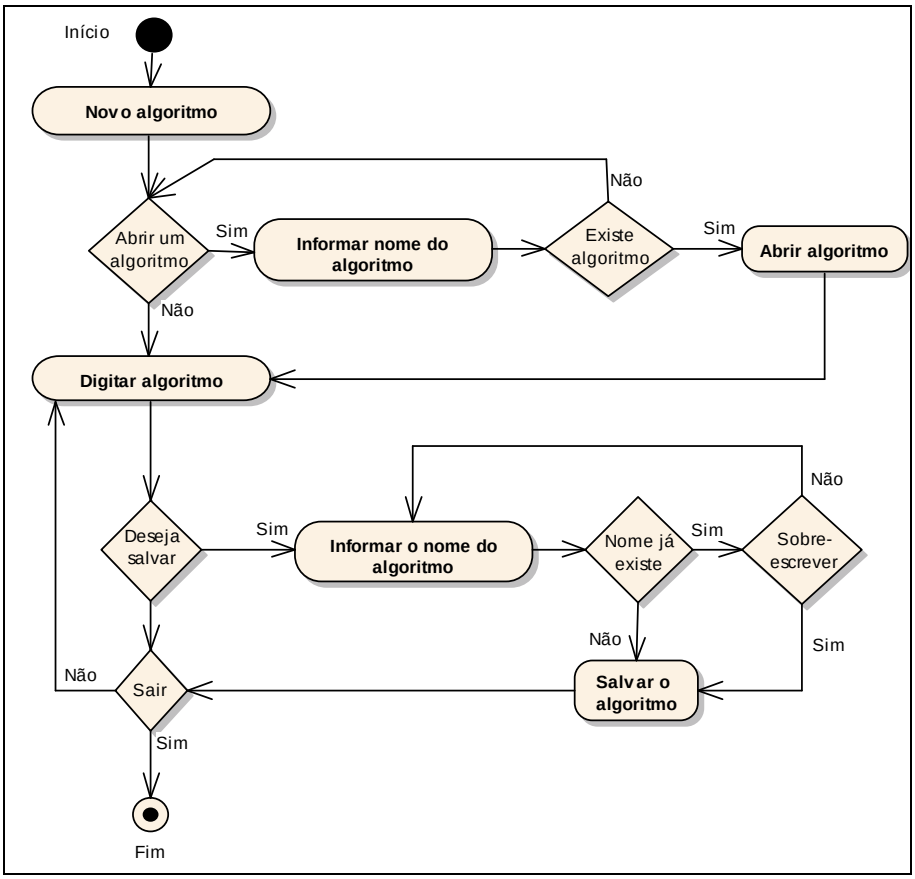
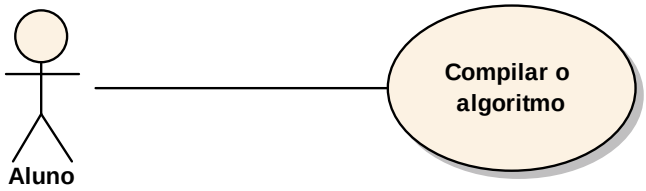


Figura 2 – Diagrama de atividades: digitar o algoritmo

O processo de compilação do algoritmo verifica se não existem erros léxicos, sintáticos ou semânticos no algoritmo digitado. Para realizar este processo o usuário deve possuir um algoritmo digitado. O quadro 17 e a figura 3 apresentam o caso de uso e o diagrama de atividades para a compilação do algoritmo.



UC.2. Compilar o algoritmo: verificar se o algoritmo digitado não possui erros léxicos, sintáticos ou semânticos, gerando código intermediário.	
Pré-condições	Deve haver um algoritmo digitado no editor da ferramenta.
Fluxo principal	1. Selecionar a opção **Compilar.** 2. A ferramenta faz a análise do algoritmo informando se foram encontrados erros léxicos, sintáticos ou semânticos. 3. É gerado um código intermediário que será utilizado para a execução.
Fluxo alternativo	Não há .
Fluxo de exceção	1. Se não houver um algoritmo digitado, ocorre um erro sintático (início do algoritmo esperado). 2. A geração do código intermediário e a habilitação da opção para executar o algoritmo somente serão realizadas se não houver nenhum erro léxico, sintático ou semântico.
Pós-condições	Será habilitada a opção **Executar**.
Requisitos atendidos	1. Compilar algoritmos escritos em uma linguagem de programação estruturada e em português. 2. Efetuar as análises léxica, sintática e semântica do algoritmo digitado, informando os possíveis erros detectados. 3. Gerar código intermediário que possa ser executado.

Quadro 17 – Caso de uso: compilar o algoritmo

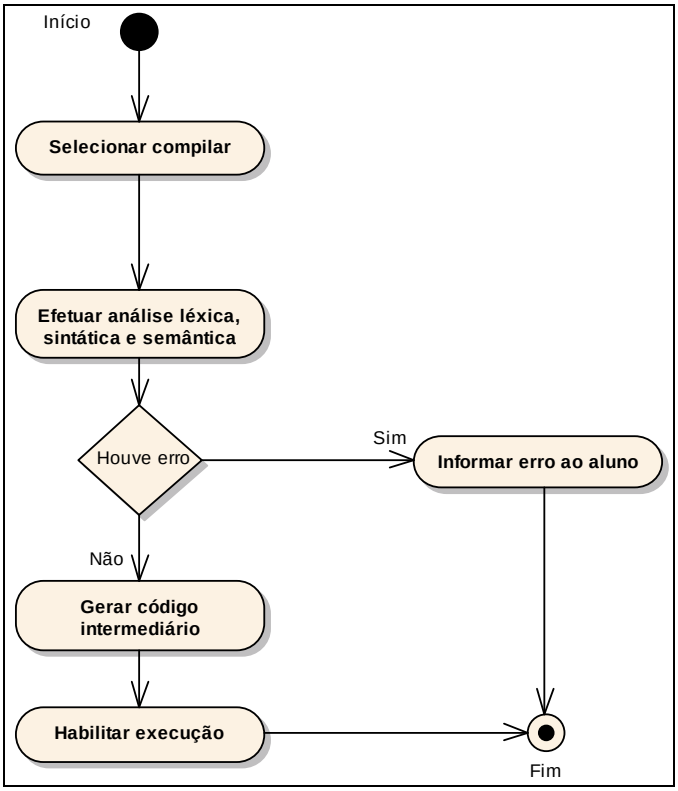
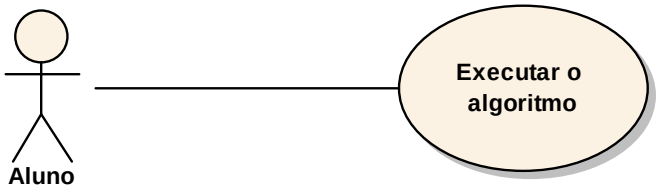


Figura 3 - Diagrama de atividades: compilar o algoritmo

O processo de execução do algoritmo executa o algoritmo compilado, com base no código intermediário gerado. Para realizar este processo o usuário possui duas opções:

executar o algoritmo até o fim ou executar o algoritmo passo a passo, visualizando o valor das variáveis declaradas. O quadro 18 apresenta o caso de uso e a figura 4 o diagrama de atividades para a execução do algoritmo.



UC.3. Executar o algoritmo: executar o algoritmo compilado com base no código intermediário gerado.	
Pré-condições	O algoritmo deve ter sido compilado e não possuir erros léxicos, sintáticos ou semânticos.
Fluxo principal	1. Selecionar a opção **Executar**. 2. A ferramenta abre outra janela. Nela o aluno visualiza o código do algoritmo, as variáveis do algoritmo juntamente com seu valor atual e uma área para entrada e saída de dados. 3. Selecionar a forma de execução: 3.1 Se optar pela execução até o fim, o algoritmo será executado, parando somente quando necessitar de uma entrada via teclado. 3.2 Se optar pela execução passo a passo, o algoritmo será executado linha por linha, sendo que após a execução de cada linha, o valor das variáveis é atualizado. 4. Ao final da execução é apresentada a mensagem “FIM DE EXECUÇÃO”
Fluxo alternativo	Não há.
Fluxo de exceção	1. Caso ocorra algum erro durante a execução, o erro será informado ao aluno e a execução cancelada.
Pós-condições	O aluno saberá se seu algoritmo funciona adequadamente.
Requisitos atendidos	1. Permitir que o código intermediário gerado possa ser executado. 2. Permitir que a execução do algoritmo seja feita passo a passo, isto é, linha por linha. 3. Ter uma área para acompanhar o valor das variáveis durante a execução passo a passo do algoritmo.

Quadro 18 – Caso de uso: executar o algoritmo

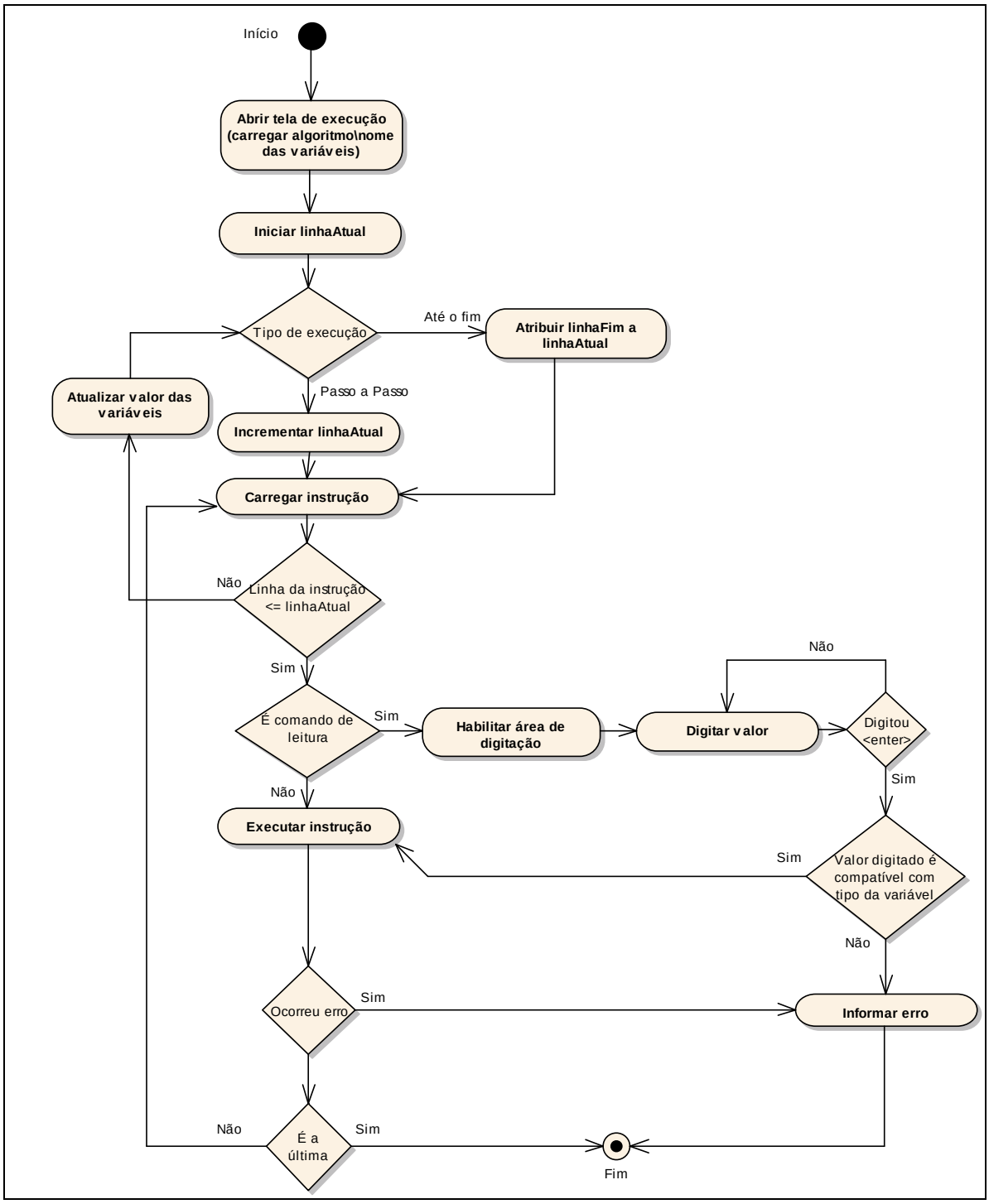


Figura 4 – Diagrama de atividades: executar o algoritmo

3.2.2.2 Diagrama de classes

Para a especificação da ferramenta, foram modeladas as classes representadas na figura 5.

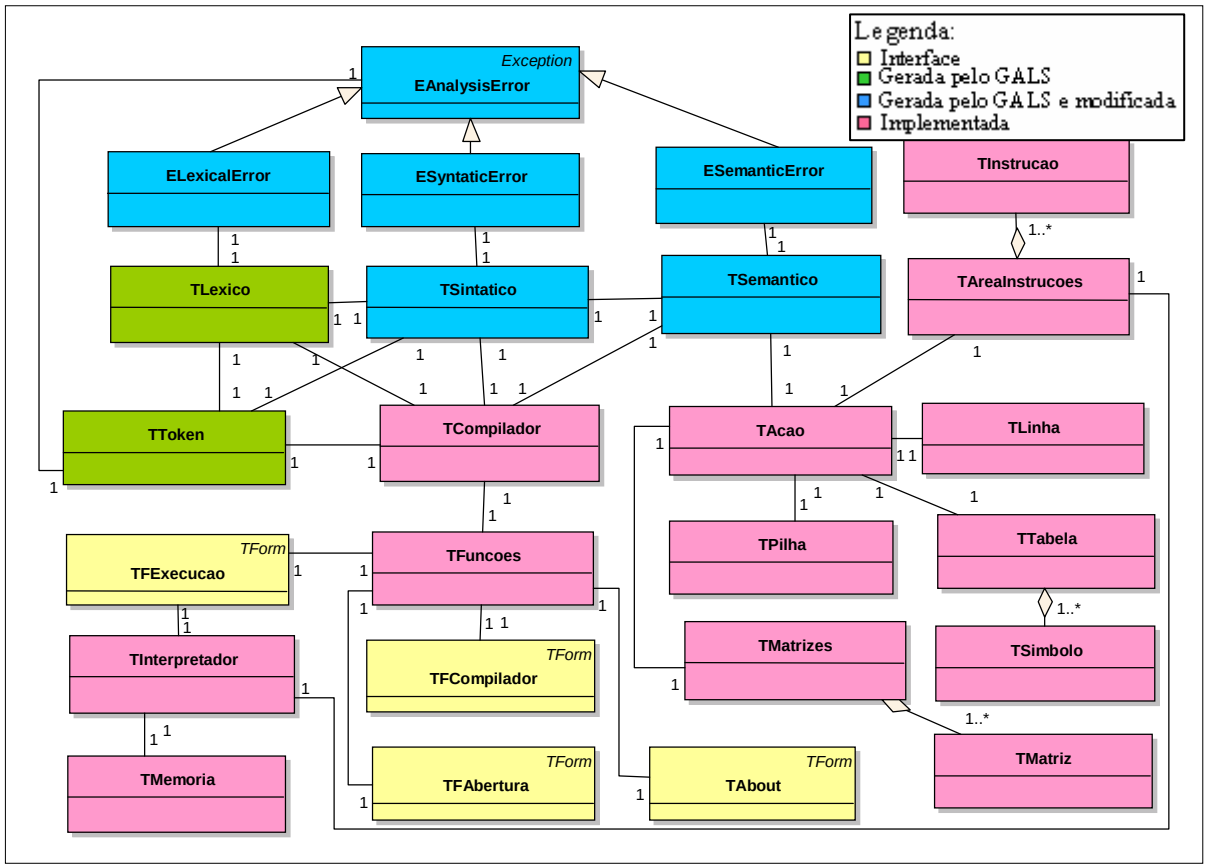


Figura 5 – Diagrama de classes

As classes `TFAbertura`, `TFExecucao`, `TFCompilador` e `TAbout` são as classes de interface, sendo que a classe `TFCompilador` está associada à classe `TFuncoes`, onde estão os métodos de manipulação das funções de controle da ferramenta.

A classe `TCompilador` é a classe principal do sistema. Possui como atributos os analisadores léxico (`TLexico`), sintático (`TSintatico`) e semântico (`TSemantico`), além de um atributo para armazenar uma possível mensagem de erro (`msgDeErro`) e outro que indica o *token* que contém o erro (`tokenErrado`). `TSintatico` controla todo o processo de compilação, chamando os métodos de `TLexico` quando um *token* é necessário e de `TSemantico` para a verificação da semântica estática e geração de código. Nesse caso, `TSemantico` chama os métodos da classe `TAcoes` passando o *token* a ser analisado. Caso ocorra algum erro durante a compilação de um algoritmo, as classes `ELexicalError`, `ESyntaticError` e `ESemanticError` são chamadas pelas classes `TLexico`, `TSintatico` e `TSemantico`, respectivamente, retornando o erro para a classe `TCompilador` que irá formatar

a mensagem e informar o usuário. A classe `TCompilador` é chamada pela classe `TFuncoes`, através do método `chamada`, que recebe o algoritmo como parâmetro. As classes descritas são apresentadas de forma detalhada na figura 6.

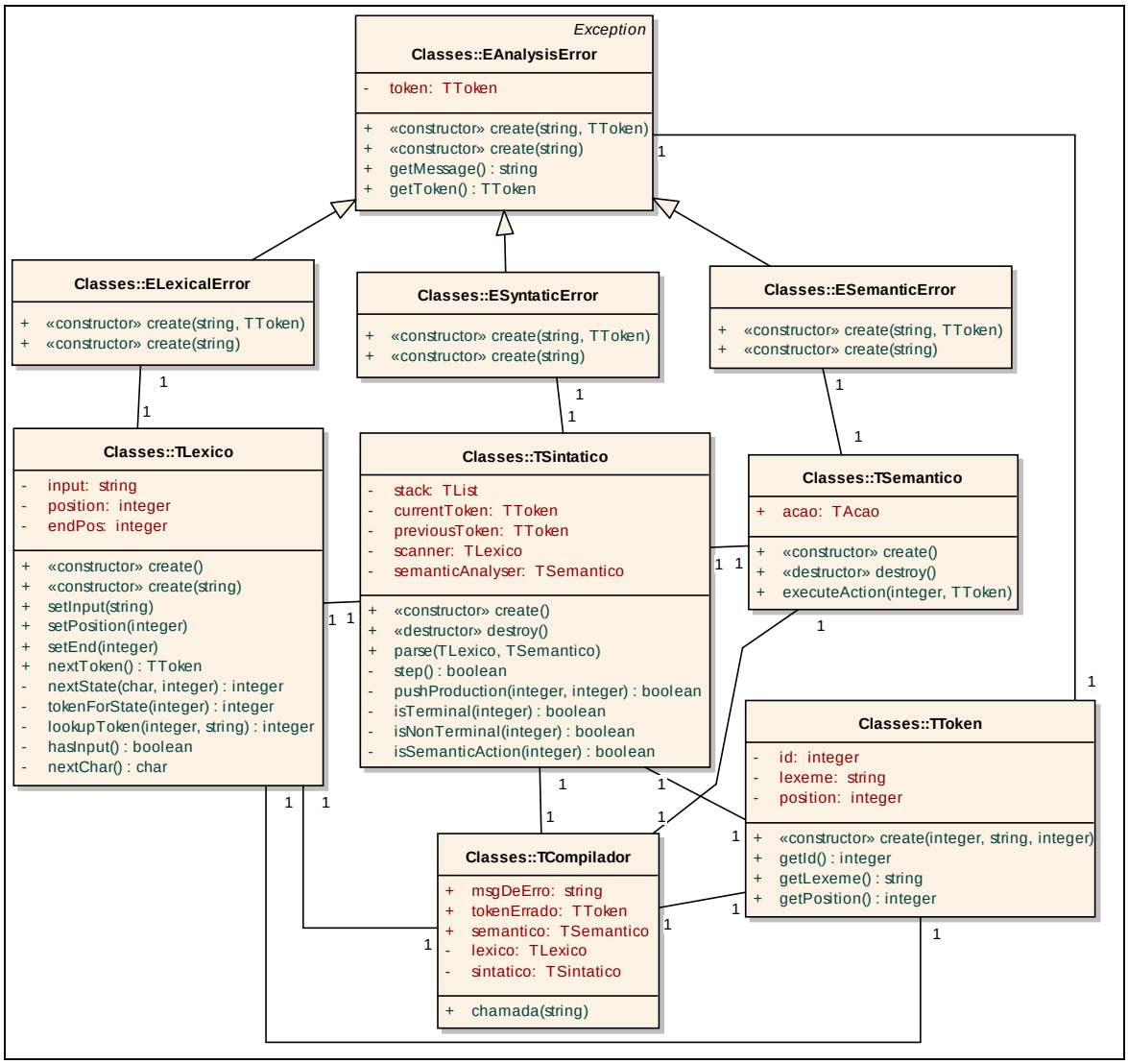


Figura 6 – Classes da compilação

A classe `TAcao` é responsável pela execução das ações semânticas e está associada às classes `TLinha`, `TAreaInstrucoes`, `TPilha`, `TMatrizes` e `TTabela`. Possui os seguintes atributos:

a) `instrucoes`: é um objeto da classe `TAreaInstrucoes`. Sua função é armazenar as instruções, ou seja, o código intermediário gerado. A classe `TAreaInstrucoes` possui um *array* de objetos de `TInstrucao`, sendo que uma instrução contém o

número da instrução, o código[6], o parâmetro e a linha da instrução no algoritmo. Possui métodos para inclusão, alteração e busca de instruções;

b) `tabela`: é um objeto da classe `TTabela`, cuja função é armazenar os símbolos ou identificadores do algoritmo. A classe `TTabela` possui um *array* de objetos de `TSimbolo`, sendo que cada símbolo possui um nome cujo valor é o identificador, uma categoria que indica o tipo[7] do identificador, um atributo cujo valor é seu endereço na memória, se ele foi inicializado e se ele foi utilizado. Possui métodos para inclusão, alteração e busca de símbolos;

c) `tipo`: é um objeto da classe `TMatrizes` que armazena os tipos declarados. A classe `TMatrizes` possui um *array* que armazena objetos de `TMatriz`. Possui métodos para inclusão, alteração e busca de matrizes. Um objeto da classe `TMatriz` contém o nome da matriz, o início e o fim da matriz, a categoria e um *array* contendo os nomes das variáveis declaradas daquele tipo bem como o endereço inicial na memória;

d) `linha`: é um objeto do tipo `TLinha` que tem por função armazenar as posições de início e fim de cada linha do algoritmo. Possui o método `localizar`, que retorna a linha com base na posição passada como parâmetro;

e) `verificacaoTipos`: é um *array* utilizado para fazer a verificação de compatibilidade de tipos, isto é, verificar a coerência entre declaração e uso dos identificadores;

f) `pilha`: objeto de `TPilha`, que armazena endereços. A classe `TPilha` possui um *array* de inteiros que armazena os endereços não resolvidos, utilizados em instruções de desvio (JMF, JMT e JMP) em comandos de seleção e repetição.

[6] As instruções são apresentadas no apêndice C.
[7] Os tipos possíveis são: inteiro, real, lógico, cadeia, caractere e matriz.

Possui métodos para empilhar e desempilhar valores;

g) `contexto`: informa se um identificador está sendo declarado como um tipo ou como uma variável;

h) `vt`: armazena o número total de variáveis declaradas no algoritmo;

i) `vp`: é usado para contar o número de variáveis de determinado tipo;

j) `ponteiro`: indica o número da próxima instrução que será gerada;

k) `tupla`: é um *array* que armazena símbolos cujas informações são necessárias para a execução de uma ação semântica;

l) `sinal`: é usado para indicar se uma expressão foi precedida por um sinal unário (+ ou -);

m) `desvio`: informa qual o nível do encadeamento que está sendo analisado em comandos escolha-caso encadeados.

A classe `TAcao` possui um método para a geração das instruções (`gerarInstrucao`), um método que recebe um número como parâmetro e retorna a *string* com o nome do tipo por extenso (`nomeEquivalente`) e 75 métodos que são as ações semânticas.

A figura 7 apresenta as classes acima descritas.

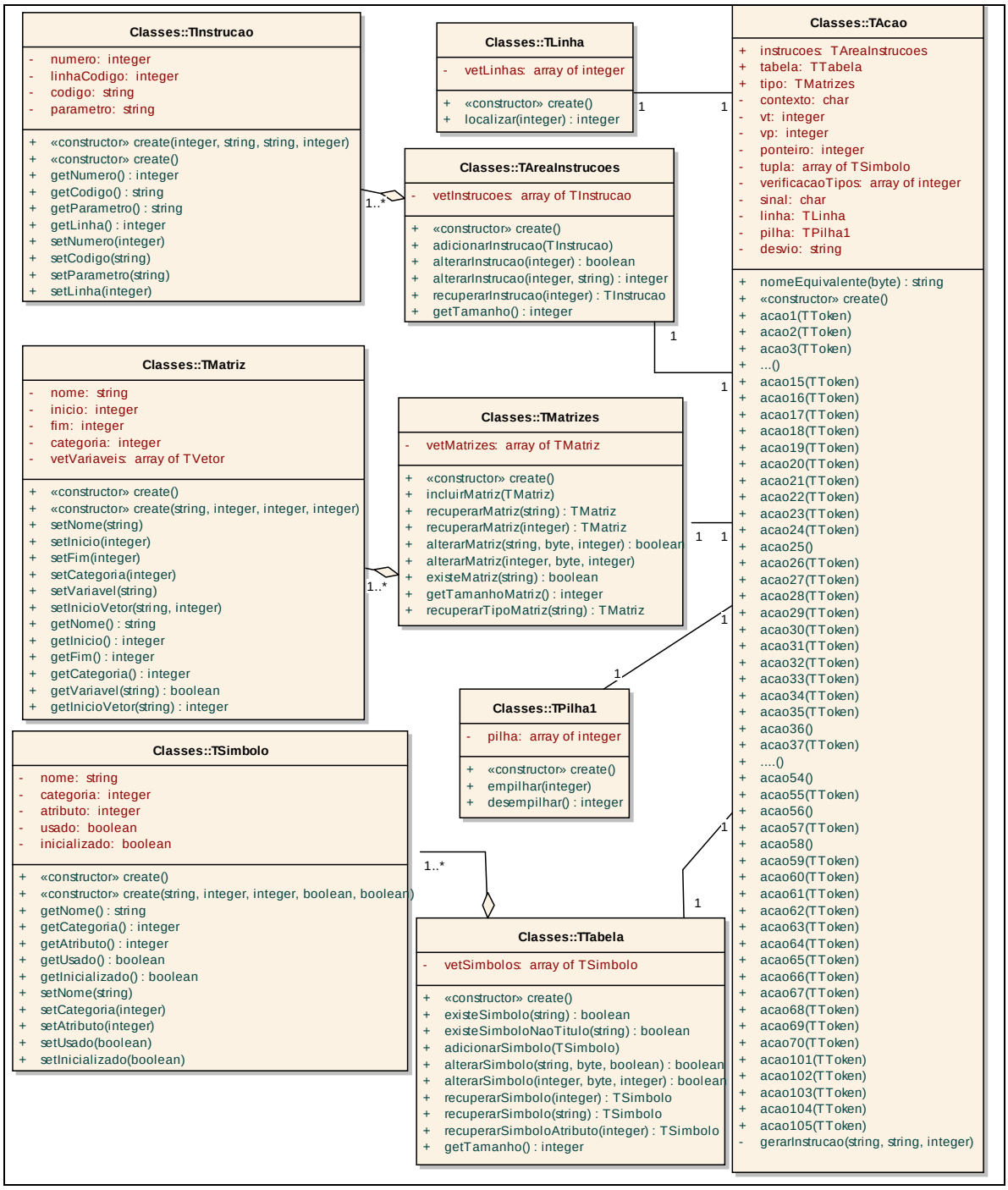


Figura 7 – Classes da análise semântica

Após a compilação, com base no código gerado, pode ser feita a execução do algoritmo. Quando a execução é chamada, um objeto da classe `TInterpretador` é instanciado para que o código intermediário gerado possa ser executado.

A classe `TInterpretador` possui o atributo `memoria`, um objeto do tipo `TMemoria`, que tem a função de armazenar os valores na memória durante a execução do algoritmo. Tem

também o atributo `ponteiro` que indica a próxima instrução a ser executada. Possui um método para chamar a próxima instrução a ser executada (`proximaInstrucao`), um método para chamar a instrução de leitura após um valor ter sido digitado pelo aluno (`chamada_rea`), um método para reiniciar a execução de um algoritmo, um método que recebe dois tipos e verifica se são compatíveis (`verificarCompatibilidade`) e 41 métodos que são as instruções da linguagem intermediária. A classe `TMemoria` possui um *array* de *string* onde são armazenados os valores manipulados pelo algoritmo e uma variável que armazena o topo da memória, que funciona como uma pilha. Possui métodos para alocar, desalocar e alterar os valores da memória.

As classes descritas são apresentadas na figura 8.

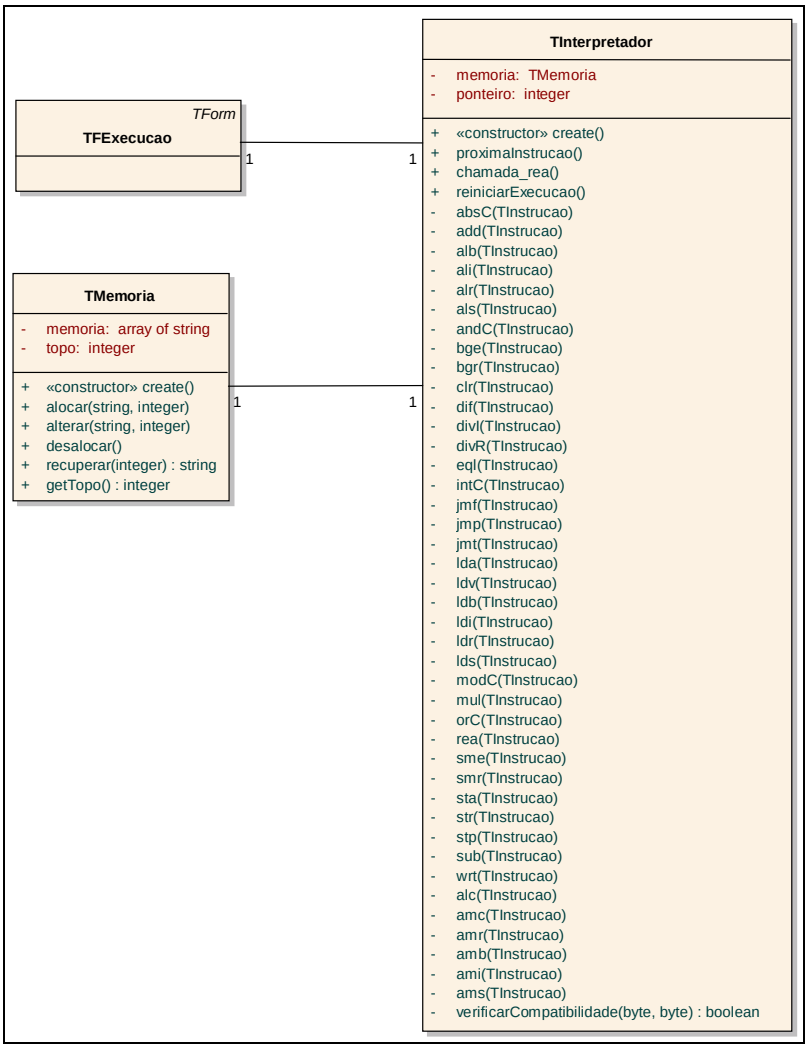


Figura 8 – Classes da execução

3.3 IMPLEMENTAÇÃO

Nesta seção são apresentados os aspectos sobre a implementação do protótipo e as ferramentas utilizadas para a construção.

3.3.1 Ferramentas utilizadas

Para implementação do protótipo foi utilizado o ambiente de desenvolvimento Borland Delphi 7. A linguagem utilizada pelo ambiente é *Object Pascal*. No quadro 19 é apresentando um trecho do código. Este método é responsável por efetuar a compilação de um algoritmo e exibir os erros detectados.

```
procedure TCompilador.chamada (texto : string);
...
begin
  lexico := TLexico.create;
  sintatico := TSintatico.create;
  semantico := TSemantico.create;
  lexico.setInput(texto);
  try
    sintatico.parse(lexico,semantico);
    ...
  except
    on e:ELexicalError do
    begin
      msgDeErro:='   -   '+ ELexicalError(e).getMessage+'   ';
      tokenErrado:=ELexicalError(e).getToken;
    end;
    on e:ESyntaticError do
    begin
      msgDeErro:='era esperado  '+ ESyntaticError(e).getMessage;
      tokenErrado:=ESyntaticError(e).getToken;
    end;
    on e:ESemanticError do
    begin
      msgDeErro:=ESemanticError(e).getMessage;
      tokenErrado:=TToken.create(0,'',0);
    end;
  end;
...
end;
```

Quadro 19 – Método chamada

Na implementação também foi utilizado o GALS, uma ferramenta para geração de analisadores léxicos e sintáticos. GALS é a abreviatura de Gerador de Analisadores Léxicos e Sintáticos. É uma ferramenta *freeware* que, com base em definições regulares e uma

gramática, gera os analisadores para três linguagens (Java, C++ ou Delphi). Tem a opção de gerar o analisador léxico, o analisador sintático ou ambos (GESSER, 2003, p.39). Esta ferramenta pode gerar analisadores sintáticos descendentes ou analisadores sintáticos redutivos.

3.3.2 Implementação da ferramenta

A compilação de um algoritmo inicia no código apresentado no quadro 19. Durante a execução deste código, pode ser encontrado um erro. Caso isto ocorra, o fonte apresentado no quadro 20 é executado.

```
ERRO LÉXICO
  raise ELexicalError.create(    SCANNER_ERROR[oldState],
                                 TToken.create(0,copy(input,start,1),start)   );

ERRO SINTÁTICO
  raise ESyntaticError.create(   PARSER_ERROR[x], currentToken   );

ERRO SEMÂNTICO
  raise ESemanticError.create(   'ERRO: Na linha' +
                                 inttostr(linha.localizar(token.getPosition)) +
                                 '  ,' + ' variável "' + token.getLexeme +
                                 '" não declarada ou é o nome do algoritmo.'  );
```

Quadro 20 – Detecção de erros durante a análise

Nota-se que os parâmetros dos erros são diferentes. Isto deve-se ao fato de que nas análises léxica e sintática as mensagens de erro estão armazenadas em constantes do tipo *array*, quais sejam `SCANNER_ERROR` e `PARSER_ERROR`. Então, quando da ocorrência de um erro, é passada a posição onde está a mensagem de erro e o *token* que causou o erro. Já na análise semântica, não é utilizada nenhuma constante. Por este motivo a mensagem já é passada pronta.

Durante a análise sintática, o analisador léxico é chamado quando um *token* é necessário. E a cada *token* analisado, o analisador sintático verifica se existe uma ação semântica associada a ele. Caso exista, é chamado o analisador semântico, isto é, o código do quadro 21 é executado.

```
...
semanticAnalyser.executeAction(x-FIRST_SEMANTIC_ACTION, previousToken);
...
```

Quadro 21 – Reconhecimento de ação semântica

O primeiro parâmetro corresponde ao número da ação que deve ser executada e o segundo é o *token* associado à ação. Na classe `TSemantico` é chamada a ação correspondente ao número passado como parâmetro, conforme mostrado no quadro 22.

```
procedure TSemantico.executeAction(action : integer; const token : TToken);
begin
case action of
  1 : acao.acao1(token);
  2 : acao.acao2(token);
  3 : acao.acao3(token);
...
end;
```

Quadro 22 – Execução da ação semântica

Foram implementadas as 75 ações semânticas apresentadas no apêndice B. Nos quadros 23, 24, 25, 26, 27, 28 e 29 podem ser vistos exemplos de ações semânticas.

A ação #1 apresentada no quadro 23, é chamada quando é identificado o nome do algoritmo. Ela cria um objeto de `TSimbolo`, contendo o nome do algoritmo e inclui este símbolo na tabela de símbolos.

```
procedure TAcao.acao1(token: TToken);
...
begin
  simbolo:=TSimbolo.create(token.getLexeme, 999, vt, true, true);
  tabela.adicionarSimbolo(simbolo);
end;
```

Quadro 23 – Ação semântica #1

No quadro 24 é mostrada a implementação da ação #3. Ela é executada quando é reconhecido o fim do algoritmo. Gera a instrução STP e verifica se as variáveis declaradas foram utilizadas e inicializadas. Caso alguma variável não tenha sido usada e/ou inicializada, isto é informado ao aluno.

```
procedure TAcao.acao3(token : TToken);
...
begin
  gerarInstrucao('STP','0', token.getPosition);
  for ind:= 0 to tabela.getTamanho-1 do begin
    if tabela.recuperarSimbolo(ind).getUsado <> true then
      aviso:=  'AVISO: A variável "' + tabela.recuperarSimbolo(ind).getNome +
               '" não foi utilizada. Ela é necessária?'
    else
    if (tabela.recuperarSimbolo(ind).getInicializado <> true) then begin
      aviso:=  'AVISO: A variável "' + tabela.recuperarSimbolo(ind).getNome +
               '" não foi inicializada.';
      ...
    end;
  end;
end;
```

Quadro 24 – Ação semântica #3

A ação #21 apresentada no quadro 25 é chamada quando é reconhecido um comando de saída. Ela gera a instrução WRT .

```
procedure TAcao.acao21(token : TToken);
begin
  gerarInstrucao('WRT','0', token.getPosition);
end;
```

Quadro 25 –Ação semântica #21

O quadro 26 apresenta a implementação da ação #22 que é executada quando uma constante literal é reconhecida no algoritmo. Ela gera a instrução LDS passando como parâmetro a constante reconhecida.

```
procedure TAcao.acao22 (token : TToken);
begin
  gerarInstrucao('LDS',token.getLexeme, token.getPosition);
end;
```

Quadro 26 –Ação semântica #22

Quando é reconhecida uma variável do tipo matriz em um comando de saída a ação #23 é chamada. Ela gera a instrução LDA, passando como parâmetro o endereço da variável (informação recuperada na ação #102 e armazenada em `tupla`) e altera na tabela de símbolos a propriedade “usado” da variável para verdadeiro. A informação recuperada na ação #102 não é mais necessária, portanto a última posição da `tupla` é eliminada. A implementação desta ação é apresentada no quadro 27.

```
procedure TAcao.acao23(token: TToken);
begin
  gerarInstrucao('LDA',
                 inttostr(tupla[high(tupla)].getAtributo), token.getPosition);
  tabela.alterarSimbolo(tupla[high(tupla)].getNome,usado,true);
  setLength(tupla,length(tupla)-1);
end;
```

Quadro 27 – Ação semântica #23

O quadro 28 apresenta a implementação da ação #102. Ela é chamada para verificar se uma variável é do tipo matriz. Primeiramente verifica se a variável foi declarada. Em caso afirmativo, armazena em `tupla` as informações associadas à variável, que serão usadas quando da execução de outra ação semântica (ação #23). Caso contrário, gera um erro semântico.

```
procedure TAcao.acao102(token: TToken);
begin
  if (tabela.existeSimboloNaoTitulo(token.getLexeme))then begin
    if (tabela.recuperarSimbolo(token.getLexeme).getCategoria >=12) then begin
      setLength(tupla,length(tupla)+1);
      tupla[high(tupla)]:=tabela.recuperarSimbolo(token.getLexeme);
    end
    else
      raise ESemanticError.create('ERRO: Na linha  ' +
                                  inttostr(linha.localizar(token.getPosition)) +
                                  '  , ' + 'variável "' + token.getLexeme +
                                  '" não é do tipo MATRIZ.');
  end
  else
    raise ESemanticError.create('ERRO: Na linha  ' +
                                inttostr(linha.localizar(token.getPosition)) +
                                '  , ' + 'variável "' + token.getLexeme +
                                '" não declarada.');
end;
```

Quadro 28 – Ação semântica #102

Quando é preciso garantir que o valor de uma expressão é do tipo inteiro, a ação #103 (quadro 29) é chamada. Para tanto, verifica se o valor armazenado na última posição do *array* `verificacaoTipos` é igual a 2 (tipo inteiro). Caso não seja, é gerada uma mensagem de erro e a compilação é cancela. A última posição de `verificacaoTipos` é excluída.

```
procedure TAcao.acao103(token: TToken);
begin
  if ((verificacaoTipos[high(verificacaoTipos)]<>2)) then
    raise ESemanticError.create(
       'ERRO: Na linha  ' +
       inttostr(linha.localizar(token.getPosition)) +
       '  , incompatibilidade de tipos, era esperado um valor do tipo  INTEIRO  e
        foi encontrado um valor do tipo  ' +
       nomeEquivalente(verificacaoTipos[high(verificacaoTipos)]) + '  .');
  setLength(verificacaoTipos,length(verificacaoTipos)-1);
end;
```

Quadro 29 – Ação semântica #103

Conforme pode ser observado nos quadros anteriores, algumas ações geram instruções, enquanto outras apenas são responsáveis por efetivar verificações semânticas. Com o código intermediário gerado, é habilitada a execução do algoritmo, com a opção de fazê-la passo a passo. Caso a opção passo a passo seja selecionada, a cada passo serão executadas as instruções associadas à linha atual do algoritmo. Ao final da execução da linha, o valor das variáveis é atualizado. Se a opção passo a passo não for selecionada, o valor das variáveis é atualizado apenas após a execução de todas as instruções do algoritmo. Caso ocorra algum erro durante a execução, o erro é informado ao usuário e a execução é cancelada. Os quadros 30, 31, 32 e 33 apresentam o código fonte de algumas instruções.

```
procedure TInterpretador.lds(instrucao: TInstrucao);
...
begin
  try
    constante_sem_aspas:=instrucao.getParametro;
    memoria.alocar(constante_sem_aspas,memoria.getTopo);
    ponteiro:=ponteiro+1;
    proximaInstrucao;
  except
    funcao.imprimirTela('ERRO DURANTE A EXECUÇÃO: Impossível carregar constante
                         do tipo CADEIA.');
  end;
end;
```

Quadro 30 – Instrução LDS

A instrução LDS aloca uma posição na memória para armazenar a constante passada como parâmetro. Incrementa o ponteiro e chama a próxima instrução.

```
procedure TInterpretador.wrt(instrucao: TInstrucao);
...
begin
  try
    constante_sem_aspas:=memoria.recuperar(memoria.getTopo);
    if qtdLinhas < 6000 then
       funcao.imprimirTela(escreva);
    else
       showmessage('ERRO DURANTE A EXECUÇÃO: Estouro do limite de memória');
    memoria.desalocar;
    ponteiro:=ponteiro+1;
    proximaInstrucao;
  except
    showmessage('ERRO DURANTE A EXECUÇÃO: Erro na escrita, não é possível
                 escrever um valor vazio.');
  end;
end;
```

Quadro 31 – Instrução WRT

Quando a instrução WRT é executada, primeiro é verificado se não existem mais de

6000 linhas já impressas no *console*. Se isto acontecer, a execução é encerrada por estouro de memória. Caso contrário, o valor do topo da memória é escrito na tela e é liberada a posição de memória que armazena esse valor. Incrementa o ponteiro e chama a próxima instrução.

```
  procedure TCodigo.stp(instrucao: TInstrucao);
  begin
    fim:=true;
    função.imprimirTela('FIM DA EXECUÇÃO.');
  end;
```

Quadro 32 –Instrução STP

Quando é reconhecida a instrução STP, a execução do algoritmo é encerrada e é escrito na tela FIM DA EXECUÇÃO.

```
  procedure TInterpretador.lda(instrucao: TInstrucao);
  ...
  begin
    try
    s:=TSimbolo.Create;
    s:=tabela.recuperarSimboloAtributo(strtoint(instrucao.getParametro));
    tamanho:= tipo.recuperarTipoMatriz(s.getNome).getFim -
tipo.recuperarTipoMatriz(s.getNome).getInicio;
    inicio:= tipo.recuperarTipoMatriz(s.getNome).getInicio;
    indice:= memoria.recuperar(memoria.getTopo);
    valor:= strtoint(instrucao.getParametro) + (strtoint(indice) - inicio);
    inicioMemoria:= tipo.recuperarTipoMatriz(s.getNome).getInicioVetor(s.getNome);
    if((valor >=inicioMemoria) and (valor<=(inicioMemoria+tamanho)) then
    begin
      memoria.alterar(memoria.recuperar(valor),memoria.getTopo);
      ponteiro:=ponteiro+1;
      proximaInstrucao;
    end
    else
    begin
      funcao.imprimirTela ('ERRO DURANTE A EXECUÇÃO: Impossível armazenar valor da
                            variável, não existe esta posição.');
      fim:=true;
    end;
    except
      funcao.imprimirTela ('ERRO DURANTE A EXECUÇÃO: Impossível acessar o valor da
                            variável, não existe esta posição.');
    end;
  end;
```

Quadro 33 – Instrução LDA

A instrução LDA carrega o valor de uma variável do tipo matriz na memória. A primeira etapa consiste em calcular o endereço da variável na memória. Com base na posição inicial da matriz, é calculado o endereço do índice. Após é verificado se o índice está dentro da faixa da matriz. Não havendo erros, o valor é carregado no topo da memória, o ponteiro é incrementado e a próxima instrução é chamada.

3.3.3 Operacionalidade da implementação

Quando a ferramenta é executada, a tela apresentada na figura 9 é exibida. Nela o aluno deve informar seu nome para ter acesso à ferramenta. Após informar o nome, o aluno pode selecionar o botão **Entrar** e fazer uso da ferramenta.

Figura 9 – Tela de abertura da ferramenta

A figura 10 apresenta o ambiente para desenvolvimento de algoritmos em Portugol. Ele é composto por um editor de textos com opções para edição de textos e manipulação de arquivos, por um compilador (opção **Compilar**) e por um interpretador (opção **Executar**).O aluno pode acessar as funções do ambiente fazendo uso dos botões e das teclas de atalho, como também dos menus. Na parte inferior da tela é apresentado o *status* atual do editor, informando a linha atual do *cursor*, o nome do algoritmo aberto e se o mesmo foi ou não modificado.

Conforme o algoritmo é digitado, as palavras reservadas são apresentadas em negrito; os números ficam na cor vermelha; os comentários em itálico e na cor verde; e as constantes do tipo cadeia na cor azul. O uso de cores visa facilitar a identificação dos comandos por parte dos alunos.

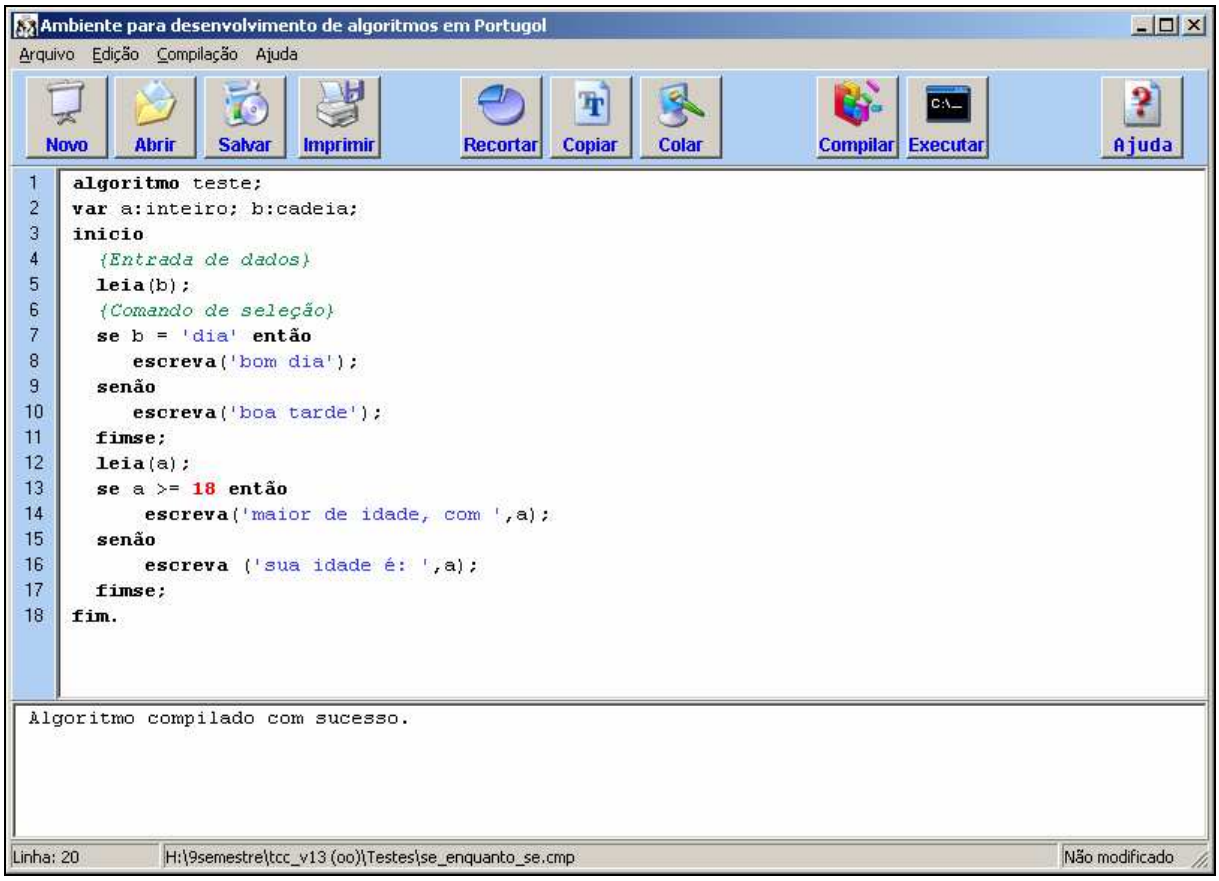


Figura 10 – Tela da ferramenta com algoritmo digitado

Quando um algoritmo é compilado, as mensagens de erro ou a informação de que o mesmo foi compilado com sucesso são apresentadas na parte inferior da tela.

Quando a execução é selecionada, uma nova tela é aberta. A figura 11 apresenta esta tela. Ela é composta por quatro áreas, sendo elas:

a) *console*: é a área, onde o aluno irá fornecer os dados quando um comando de entrada for executado e onde serão apresentados os dados quando um comando de saída for executado;

b) variáveis: contêm o nome das variáveis declaradas e seus respectivos valores. Caso a opção passo a passo seja utilizada, a cada linha executada, o valor das variáveis é atualizado;

c) algoritmo: mostra o algoritmo que será executado. Caso a opção passo a passo seja utilizada, a linha que está sendo executada no momento é destacada;

d) botões: são as opções de execução. O primeiro é para habilitar a execução passo a

passo. Nesse caso, para que a próxima linha seja executada, este botão deve ser selecionado. Se não desejar executar o algoritmo passo a passo, o botão “Executar até FIM” deve ser selecionado. Caso deseje começar a execução novamente, o botão “Executar de novo” deve ser selecionado.

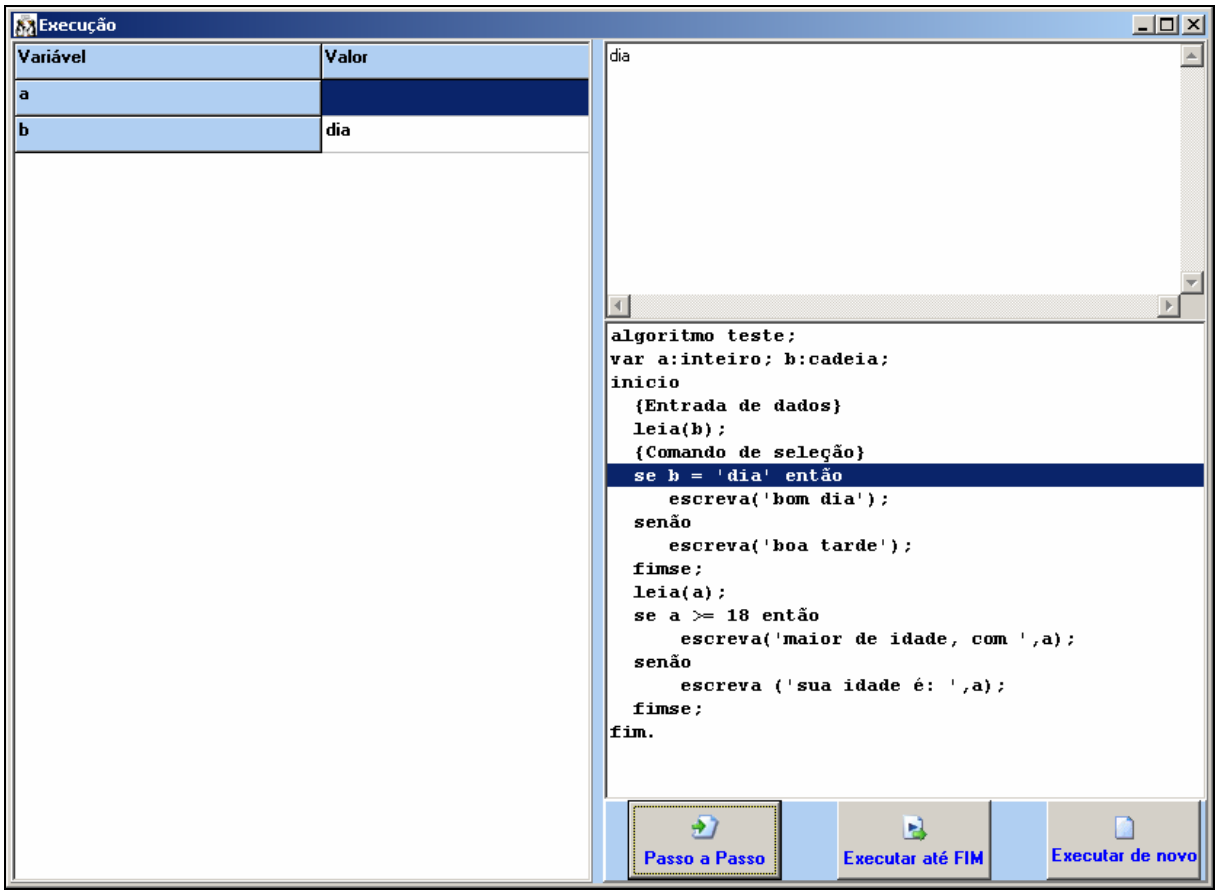


Figura 11 – Tela de execução

Para fechar a tela basta selecionar o botão com X no lado direito superior.

A ferramenta também possui uma ajuda sobre Portugol. Ela pode ser utilizada de três maneiras. A primeira delas é selecionar o botão Ajuda da tela principal (figura 10). A segunda maneira é apertar o botão F1. Em ambos os casos será aberta a tela com as opções de ajuda (figura 12). Caso o aluno deseje informações sobre um comando em particular, basta selecionar no editor do ambiente a palavra reservada referente ao comando e pressionar o botão F1. Com isto a ajuda será aberta já no tópico referente ao comando selecionado. A figura 13 apresenta o tópico de ajuda do comando de entrada de dados.

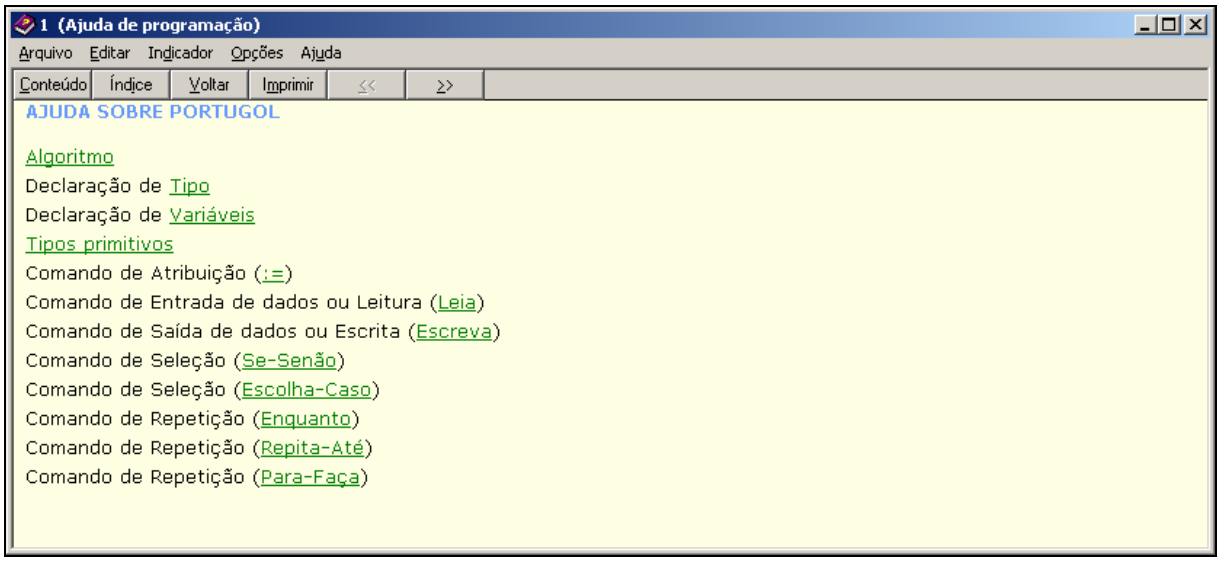


Figura 12 – Tela principal da ajuda

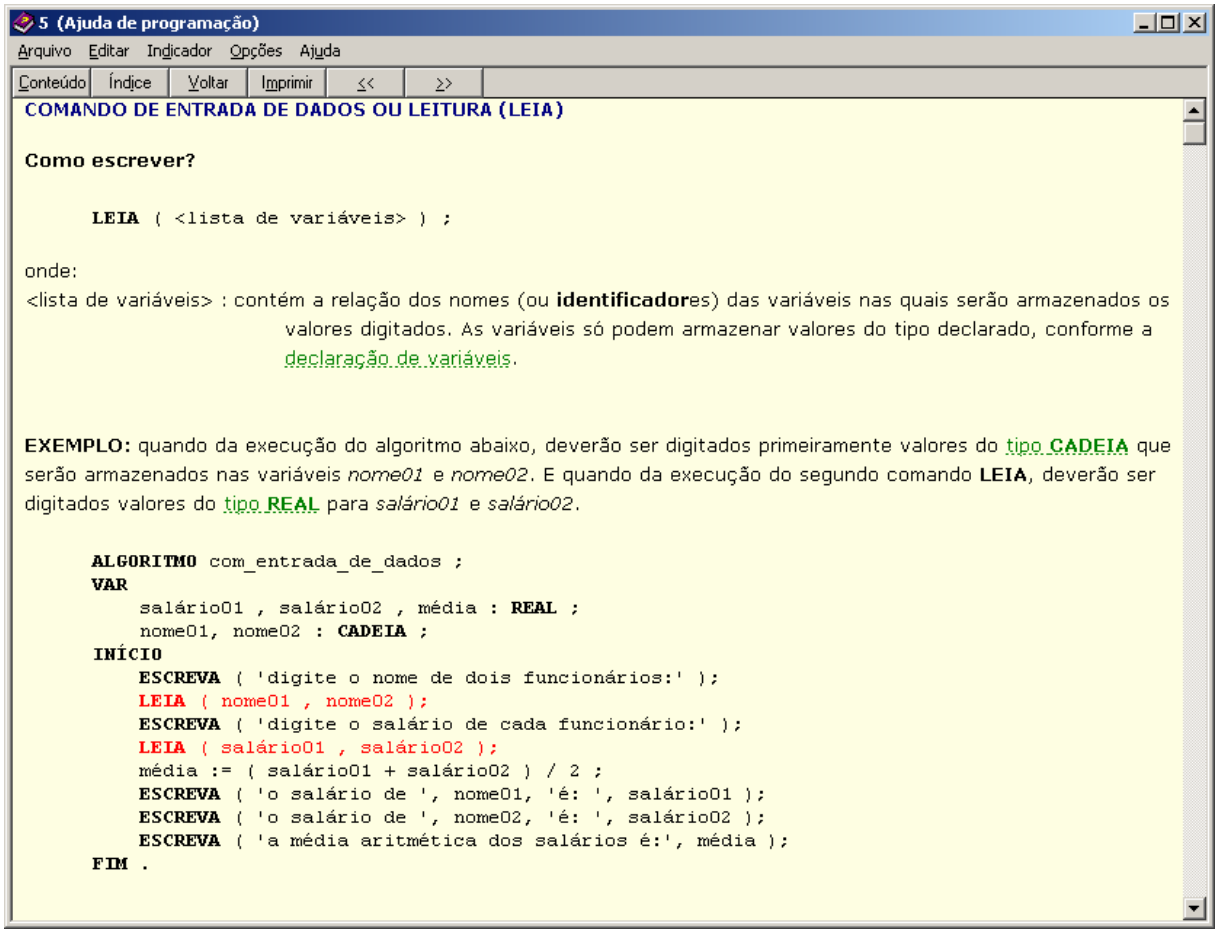


Figura 13 – Tela de ajuda do comando leia

Nas figuras anteriores, as palavras sublinhadas em verde, fornecem explicações adicionais.

Por fim, tem-se a opção **Sobre**, dentro do *menu* **Ajuda**, que contém algumas informações sobre a ferramenta (figura 14).

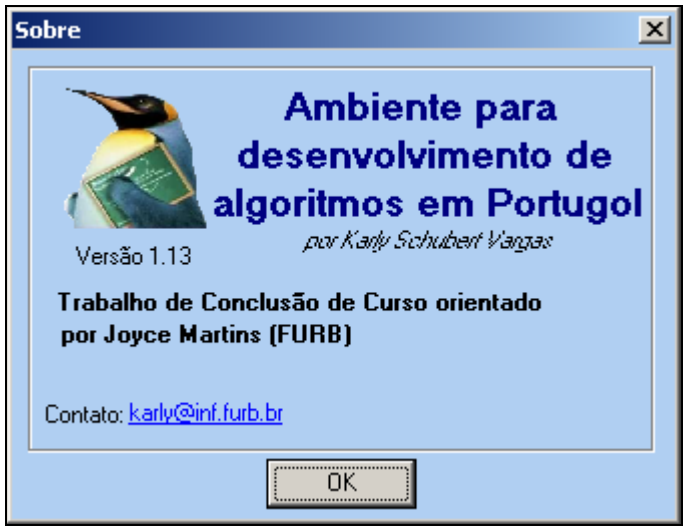


Figura 14 – Tela sobre

3.3.4 Versão para Linux

Com o objetivo de atender aos alunos que utilizam o Linux, foi disponibilizada uma versão da ferramenta para este sistema operacional. Nessa versão somente a interface foi adaptada, como pode ser visto na figura 15.

Para executar em Linux é utilizado o *wine*, um emulador de Windows dentro da ambiente Linux. Com este emulador é possível rodar programas feitos para Windows. A execução do programa em Linux é simples, basta instalar o *wine*, ir até a pasta onde foi descompactada a ferramenta e executar `wine Compilador.exe`.

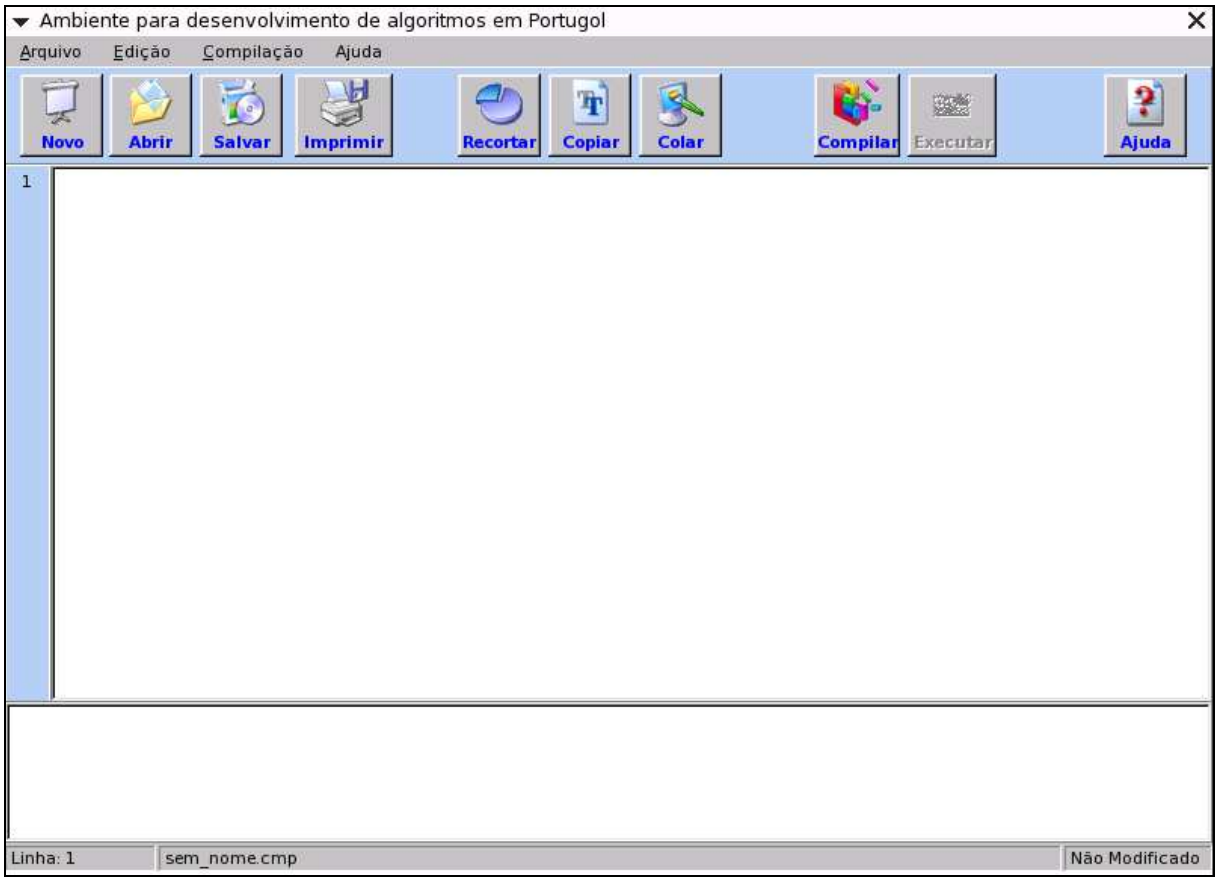


Figura 15 – Tela da ferramenta em Linux

3.3.5 Avaliação do ambiente de programação

Na seção 2.5, foi apresentada a importância da definição de critérios para avaliar a qualidade de um software. Campos (2004, p. 96-97) fez a definição de alguns itens, que devem ser seguidos para se obter um software com qualidade. Com relação à ferramenta desenvolvida, tem-se:

a) possui ajuda sensível ao contexto;

b) informa ao aluno qual o erro encontrado no algoritmo e como corrigi-lo, apresentando mensagens de erro adequadas para cada situação;

c) fornece *feedback* em todas as situações. Durante a compilação de um algoritmo a ferramenta apresenta mensagem informando a ocorrência ou não de erros. Da mesma forma, durante a execução, caso ocorra um erro, é informado ao aluno,

senão ao final da execução é apresentada a mensagem de fim de execução;

d) foi desenvolvida em conjunto com o professor da disciplina de Introdução à Programação, garantindo, com isso, que a mesma seja adequada ao nível do aluno e ao currículo do curso. Desta forma é possível integrar o ensino em sala de aula com o uso da ferramenta;

e) possui opção de execução passo a passo do algoritmo, de forma a fornecer uma visualização do algoritmo que foi escrito, apresentando ao aluno o resultado da execução de cada um dos comandos;

f) para evitar a perda de informações, antes de um algoritmo ser compilado, o mesmo é automaticamente salvo. Com isto, o aluno pode alterar o algoritmo sem medo de perdê-lo. Além disso, a ferramenta está sendo utilizada pelos alunos e até o momento não foram relatadas situações onde a mesma não responda adequadamente, incluindo algoritmos sem erros, porém com *loop* infinito;

g) faz uso de um fundo claro e informações em fonte escura, buscando facilitar a leitura da tela;

h) foi desenvolvida com o propósito de permitir que o aluno visualize o algoritmo em execução, objetivando despertar um maior interesse do aluno pela busca de soluções adequadas;

i) emprega cores para destacar as palavras reservadas, as constantes, os números e os comentários, como pode ser observado na figura 16, facilitando a visualização do algoritmo por parte do aluno. Além disso, é feito o uso de figuras nos botões para facilitar a identificação da função dos mesmos.

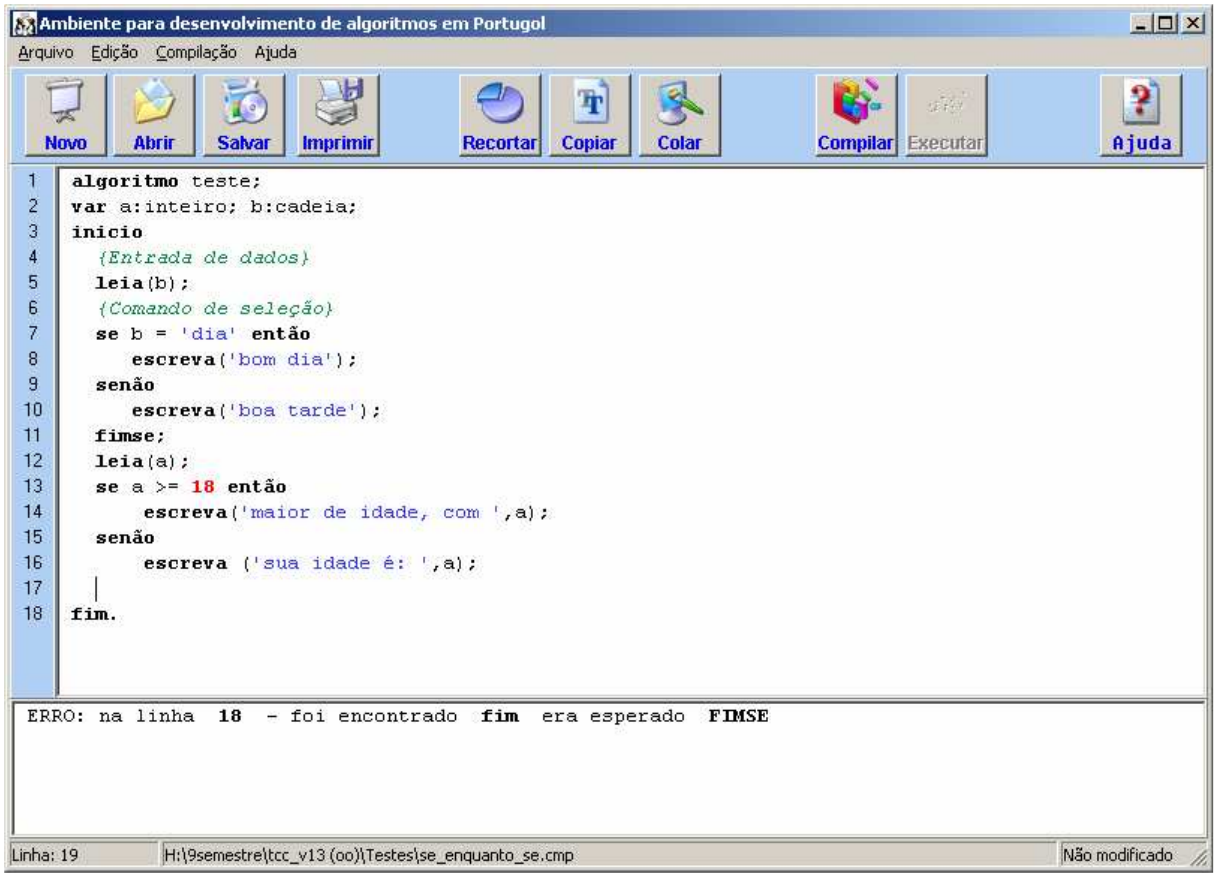


Figura 16 – Tela com mensagem de erro

3.3.6 Testes

Além dos testes feitos durante o desenvolvimento por parte da acadêmica e de sua orientadora, a ferramenta foi utilizada pelos alunos de Introdução à Programação, disciplina do curso de Ciências da Computação da FURB, nos períodos matutino e noturno. A ferramenta foi utilizada dentro e fora da sala de aula. A fase de testes inclui também a participação do professor da disciplina, de forma que erros encontrados durante o processo foram resolvidos.

Com base nos testes, os alunos responderam questionários avaliativos sobre a ferramenta, os resultados destes questionários são apresentados na seção 3.5.

3.4 RESULTADOS E DISCUSSÃO

Para avaliar a ferramenta os alunos responderam dois questionários[8], um quando conheceram a ferramenta no início do semestre e outro quando tiveram acesso à versão final. O primeiro questionário contém as seguintes perguntas:

a) no início da aula, quando você ainda não conhecia a ferramenta, foi fácil usá-la? (5-muito fácil; 4-fácil; 3-razoável; 2-difícil; 1-muito difícil);
b) no final da aula, como você classifica a facilidade de uso da ferramenta? (5-muito fácil; 4-fácil; 3-razoável; 2-difícil; 1-muito difícil);
c) a ferramenta responde de acordo com o esperado? (5-sempre; 4-quase sempre; 3-algumas vezes; 2-quase nunca; 1- nunca);
d) os ícones e atalhos são condizentes com as funções associadas? (5-sempre; 4-quase sempre; 3-algumas vezes; 2-quase nunca; 1- nunca);
e) como você classifica a interface da ferramenta? (5-muito boa; 4-boa; 3-razoável; 2-ruim; 1-não consegui utilizar);
f) as mensagens de erro da ferramenta levaram você a descobrir seu erro? (5-sempre; 4-quase sempre; 3-algumas vezes; 2-quase nunca; 1- nunca);
g) como você classifica a ajuda sobre Portugol da ferramenta? (5-muito boa; 4-boa; 3-razoável; 2-ruim; 1-não utilizei);
h) a ferramenta auxiliou no aprendizado da disciplina? (5-plenamente; 4-adequadamente;3-razoavelmente; 2-insuficientemente; 1 – não auxiliou);
i) você acha que o uso de uma ferramenta auxilia no aprendizado da disciplina? (5-Sim, está ferramenta ajuda; 4-Sim, mas talvez outra ferramenta fosse melhor; 3-Não, somente no papel seria suficiente);

[8] Disponíveis no apêndice D

j) que nota você daria para a ferramenta?

No segundo questionário somente as perguntas dos itens a e b foram alteradas para:

a) com que freqüência você utilizou a ferramenta? (5-diariamente; 4- semanalmente; 3- mensalmente; 2-somente em sala de aula; 1-não utilizei);

b) como você classifica a facilidade de uso da ferramenta? (5-muito fácil; 4-fácil; 3-razoável; 2-difícil; 1-muito difícil).

As perguntas têm 5 respostas possíveis, exceto a última que é uma pergunta aberta. O resultado do questionário pode ser visto nas tabelas 1 e 2, e gráficos comparativos no apêndice E.

NOTURNO	5		4		3		2		1	
	1º	2º	1º	2º	1º	2º	1º	2º	1º	2º
Pergunta 1	63%	7%	33%	43%	5%	20%		30%		
Pergunta 2	75%	53%	23%	23%	3%	23%				
Pergunta 3	43%	40%	48%	53%	10%	7%				
Pergunta 4	70%	77%	28%	17%	3%	7%				
Pergunta 5	50%	30%	38%	57%	10%	13%	3%			
Pergunta 6	48%	33%	33%	40%	18%	23%	3%	3%		
Pergunta 7	45%	23%	28%	50%	5%	10%	8%		15%	17%
Pergunta 8	60%	57%	40%	33%		7%		3%		
Pergunta 9	93%	93%	8%	7%						
Pergunta 10	9,1	8,7								

Tabela 1 – Avaliação do noturno

MATUTINO	5		4		3		2		1	
	1º	2º	1º	2º	1º	2º	1º	2º	1º	2º
Pergunta 1	48%	3%	52%	48%		21%		28%		
Pergunta 2	66%	34%	34%	55%		10%				
Pergunta 3	41%	21%	55%	79%	3%					
Pergunta 4	66%	62%	24%	34%	10%	3%				
Pergunta 5	48%	31%	48%	52%	3%	14%		3%		
Pergunta 6	38%	31%	45%	45%	17%	24%				
Pergunta 7	41%	28%	48%	38%		7%			10%	28%
Pergunta 8	62%	52%	34%	34%	3%	14%				
Pergunta 9	97%	97%	3%	3%						
Pergunta 10	9,1	9								

Tabela 2 – Avaliação do matutino

A pergunta 1 do primeiro questionário tem por objetivo verificar qual foi a primeira impressão que os alunos tiveram ao usar o ambiente para desenvolvimento de algoritmos. O resultado foi bom pois praticamente 100% dos alunos acharam muito fácil ou fácil de utilizar.

Com as respostas da pergunta 1 do segundo questionário pode-se verificar que boa parte dos alunos utilizou a ferramenta fora da sala de aula, o que demonstra interesse por buscar conhecimento extra classe.

Comparando as respostas da pergunta 2, juntamente com as sugestões feitas, nota-se que os alunos começaram a solicitar que a ferramenta ofereça algumas facilidades que não estão de acordo com o objetivo da disciplina como, por exemplo, o recurso de completar automaticamente o nome de uma variável ou a estrutura sintática de um comando.

A pergunta 3 visa verificar a capacidade do ambiente para desenvolvimento de algoritmos de responder adequadamente à função solicitada. Foi possível observar que a ferramenta responde de forma adequada na maioria das vezes.

As perguntas 4 e 5 são referentes à interface da ferramenta. Na pergunta 4 observa-se que os alunos acostumaram com os atalhos e ícones, em virtude de uma melhor avaliação no segundo questionário, sem que fossem alterados os mesmos. Com base no resultado da pergunta 5, pode-se responder que a interface da ferramenta está adequada.

Um dos objetivos do ambiente para desenvolvimento de algoritmos é permitir que através das mensagens o aluno possa identificar e corrigir seu erro. A pergunta 6 mostrou que ainda existem algumas mensagens inadequadas, em função de dificuldades no entendimento das mesmas por parte dos alunos.

A ferramenta possui uma ajuda sobre Portugol que foi avaliada pela pergunta 7. A primeira consideração a ser feita é o fato de que em torno de 20% das turmas não fez uso da ajuda. A segunda é que dentre aqueles que utilizaram, a maioria classifica a ajuda como boa ou muito boa.

Com a pergunta 8 buscou-se verificar se a ferramenta foi de auxílio ao aprendizado da disciplina. Na avaliação dos alunos, ela foi uma forma de auxílio no aprendizado de algoritmos, com isto atingindo um dos objetivos do trabalho.

Para a pergunta 9, 100% dos alunos responderam que o uso de um ambiente para desenvolvimento de algoritmos auxilia no aprendizado da disciplina, sendo que mais de 90% acha que a ferramenta apresentada ajuda no aprendizado. A nota média foi 9, mostrando que a ferramenta teve uma boa aceitação por parte dos alunos.

O professor da disciplina também fez uma avaliação da ferramenta. Quanto às mudanças no ensino, em virtude do uso da ferramenta, o professor avalia que agora pode-se apresentar ao aluno a execução passo a passo do algoritmo, analisando de forma bastante clara as alterações dos conteúdos das variáveis e elementos das matrizes na "memória". Ressalta a possibilidade de ver as alterações nos conteúdo das variáveis, como a grande melhoria que a ferramenta trouxe para o ensino de algoritmos. Destaca que os alunos elogiaram a ferramenta comentando que facilitou o entendimento e a visualização da execução.

Pode-se ainda estabelecer uma relação entre os trabalhos correlatos e a ferramenta desenvolvida:

a) apresenta uma interface mais amigável, se comparado com o AMBAP, facilitando o uso, além de ser adequada ao currículo do curso de Ciências da Computação na FURB;
b) usa o Portugol para representação dos algoritmos, diferente do ASA que utiliza a representação de fluxograma. Observa-se que esta representação não é utilizada pelo professor da disciplina de Introdução à Programação;
c) permite que o aluno digite seus algoritmos, diferente do software descrito por Tagliari (1996) que trabalha somente com exemplos pré-definidos;
d) possui um tratamento de erros mais adequado informando a linha do erro, a palavra errada e uma sugestão de correção. Já o CIFluxProg informa somente a localização aproximada do erro sem definir o mesmo.

4 CONCLUSÕES

O ensino de programação é um dos grandes desafios na área de ensino de computação, visto que a dificuldade encontrada pelos alunos é bastante elevada. É essencial o desenvolvimento de ferramentas que busquem despertar o interesse do aluno, assim como facilitar o entendimento da lógica de programação. Por este motivo, foi desenvolvida a ferramenta para auxiliar no ensino de introdução à programação.

A ferramenta é um ambiente interativo de aprendizagem. É importante que este tipo de ferramenta seja desenvolvido de forma personalizada para cada curso, uma vez que as formas usadas para representar algoritmos são diversificadas. Assim, a ferramenta foi desenvolvida com base na sintaxe utilizada para construção de algoritmos pelo professor da disciplina de Introdução à Programação do curso de Ciências da Computação da FURB. Portanto, a linguagem especificada é imperativa, estruturada e em português.

Para facilitar o entendimento da lógica de programação, os algoritmos podem ser executados passo a passo com opção para verificar o conteúdo das variáveis declaradas. Além disso, foi implementada detecção de erros para fornecer um diagnóstico do(s) problema(s) encontrado(s) no algoritmo, indicando o local onde está o erro bem como uma possível solução para o mesmo. Foram tratados erros como: símbolos léxicos incorretos, construções sintáticas inválidas, variáveis declaradas e não utilizadas, variáveis não inicializadas, entre outros. E ainda, procurando facilitar o uso da ferramenta por parte do aluno, foram empregados padrões de qualidade para a construção de softwares educacionais. Com base na avaliação da ferramenta feita pelos alunos e pelo professor da disciplina, pode-se afirmar que os objetivos estabelecidos foram atendidos.

Com o estudo feito para a elaboração deste trabalho conclui-se que é importante o uso de ferramentas adequadas ao ensino de determinado assunto, pois permitem experiências que

não são possíveis em sala de aula. Especificamente, o ensino de lógica de programação deve ser independente de se utilizar esta ou aquela linguagem. Deve ser dada ênfase na construção de algoritmos, etapa tão necessária para o desenvolvimento passo a passo da solução de um problema. Nesse sentido, o uso da ferramenta desenvolvida permite que o aluno visualize o que acontece quando a solução proposta por ele é executada em um computador, o que facilita no entendimento da lógica do algoritmo.

No desenvolvimento foram utilizadas diversas ferramentas. O GALS, em virtude de sua facilidade de uso, simplificou o processo de implementação dos analisadores léxico e sintático. O uso do Borland Delphi 7.0 proporcionou a construção da interface de modo simples, auxiliando no cumprimento dos critérios de qualidade.

4.1 EXTENSÕES

Para a continuidade ao trabalho, sugere-se as seguintes extensões:

a) melhoria nos mensagens de erro e acréscimos no tratamento de erro como, por exemplo, verificação de instruções que nunca serão executadas. Seria necessário fazer um estudo com os alunos da disciplina, pois um grupo deles ainda não consegue corrigir seus erros com base nas mensagens fornecidas;
b) implementação de registros. Para auxiliar este tipo de implementação pode-se consultar os capítulos 6 e 8 de Aho, Sethi e Ullman (1995) e o capítulo 8 de Louden (2004);
c) inclusão de subrotinas (funções e procedimentos). Informações sobre o tema podem ser encontradas no capítulo 7 de Aho, Sethi e Ullman (1995), no trabalho de conclusão de curso de Tomazelli (2004) e no capítulo 8 de Louden (2004);
d) suporte a matrizes multidimensionais. Para esta implementação sugere-se a leitura

de Louden (2004, p. 427) e do capítulo 8 de Aho, Sethi e Ullman (1995);

e) desenvolvimento de uma versão totalmente para Linux, não havendo assim mais a necessidade do uso do *wine*. Como sugestão, poderia ser feita uma implementação na linguagem Java, que é suportada pelos dois sistemas operacionais. Observa-se que o GALS gera código para esta linguagem.

REFERÊNCIAS BIBLIOGRÁFICAS

ALMEIDA, Eliana S. et al. AMBAP: um ambiente de apoio ao aprendizado de programação. In: WORKSHOP SOBRE EDUCAÇÃO EM COMPUTAÇÃO, 10., 2002 , Florianópolis. **Anais...** Florianópolis: SBC, 2002. 1 CD-ROM.

AHO, Alfred V.; SETHI, Ravi; ULLMAN, Jeffrey. **Compiladores**: princípios, técnicas e ferramentas. Tradução Daniel de Ariosto Pinto. Rio de Janeiro: LTC, 1995.

BARANAUSKAS, Maria C. C. et al. Uma taxonomia para ambientes de aprendizado baseados no computador. In: VALENTE, José A. (Org.). **O computador na sociedade do conhecimento**. São Paulo: USP; Estação Palavra, 1999. p. 45-68. Disponível em: <http://www.inf.ufsc.br/~edla/mec/>. Acesso em: 23 ago. 2004.

CAMPOS, Gilda H. B. **Metodologia para avaliação da qualidade de software educacional**: diretrizes para desenvolvedores e usuários. 1994. 232 f. Tese (Doutorado em Engenharia de Produção) – Programa de Pós-graduação em Engenharia, Universidade Federal do Rio de Janeiro, Rio de Janeiro.

FARRER, Harry et al. **Pascal estruturado**. 3.ed. Rio de Janeiro: LTC, 1999.

GALVIS-PANQUEVA, Álvaro H. Software educacional multimídia: aspectos críticos no seu ciclo de vida. **Revista Brasileira de Informática na Educação**, Florianópolis, n. 1, set. 1997. Disponível em: <http://www.sbc.org.br/index.php?language=1&subject=100& content=magazine&option=content&id=24>. Acesso em: 27 ago. 2004.

GESSER, Carlos E. **GALS**: gerador de analisadores léxicos e sintáticos. 2003. 150 f. Trabalho de Conclusão de Curso (Bacharelado em Ciências da Computação) – Centro Tecnológico, Universidade Federal de Santa Catarina, Florianópolis.

GUIMARÃES, Angelo M.; LAGES, Newton A. C. **Algoritmos e estruturas de dados**. Rio de Janeiro: Livros Técnicos e Científicos, 1994.

JARGAS, Aurélio M. **Expressões regulares**: guia de consulta rápida. São Paulo: Novatec, 2001. Disponível em: < http://guia-er.sourceforge.net/>. Acesso em 24 maio 2005.

JOSÉ NETO, João. **Introdução à compilação**. Rio de Janeiro: Livros Técnicos e Científicos Editora S.A., 1987.

LISBÔA, Maria L. B. **Modelos de linguagens de programação**. [Porto Alegre], 2004. Disponível em: <http://www.inf.ufrgs.br/aulas/mlp/material.html>. Acesso em 30 set. 2004.

LOUDEN, Kenneth C. **Compiladores**: princípios e práticas. Tradução Flávio Soares Corrêa da Silva. São Paulo: Thomson Pioneira, 2004.

MINISTÉRIO DA EDUCAÇÃO. **Diretrizes curriculares de cursos da área de computação e informática**. Brasília, 1999. Disponível em: <http://www.mec.gov.br/sesu/ftp/curdiretriz/computacao/co_diretriz.rtf>. Acesso em: 1 jun. 2005.

PALAZZO, Luiz A. M. **Introdução à programação:** PROLOG. Pelotas: Editora da Universidade Católica de Pelotas, EDUCAT, 1997.

PRESSMAN, Roger S. **Engenharia de software**. 5. ed. São Paulo: McGraw-Hill, 2002.

PRICE, Ana M. A.; TOSCANI, Simão S. **Implementação de linguagens de programação**: compiladores. 2. ed. Porto Alegre: Sagra Luzzatto, 2001.

SALIBA, Walter L. C. **Técnicas de programação**: uma abordagem estruturada. São Paulo: Makron, 1994.

SANTIAGO, Rafael; DAZZI, Rudimar L. S. Ferramenta de apoio ao ensino de algoritmos. In: SEMINÁRIO DE COMPUTAÇÃO, 13., 2004, Blumenau. **Anais eletrônicos...** Blumenau: FURB, 2004. Disponível em: <http://www.inf.furb.br/seminco/2004/artigos/96-vf.pdf>. Acesso em: 27 set. 2004.

SEBESTA, Robert W. **Conceitos de linguagens de programação**. Tradução José Carlos Barbosa dos Santos. Porto Alegre: Bookman, 2000.

TAGLIARI, Alessandra. **Protótipo de um software para o auxílio ao aprendizado de algoritmos**. 1996. 79 f. Trabalho de Conclusão de Curso (Bacharelado em Ciências da Computação) - Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau, Blumenau.

TERZIAN, Paulo. **Constructor**. [S.l.], 2005. Disponível em: <http://www.games-in.com.br/dicas/constructor.htm>. Acesso em: 31 maio 2005.

TOMAZELLI, Giancarlo. **Implementação de um compilador para uma linguagem de programação com geração de código Microsoft.Net Intermediate Language**. 2004. 83 f. Trabalho de Conclusão de Curso (Bacharelado em Ciências da Computação) - Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau, Blumenau. Disponível em: <http://www.bc.furb.br/docs/MO/2004/279136_1_1.pdf>. Acesso em: 03 jun. 2005.

USP. **ASA**: ambiente de simulação e animação de algoritmos. São Paulo, 2004. Disponível em: < http://www.edsoft.futuro.usp.br/asp-bin/softprop.asp?SE=0&ID=602&SS=construtor>. Acesso em: 24 ago. 2004.

VALENTE, José A. O papel do professor no ambiente Logo. In: VALENTE, José A. (Org.). **O professor no ambiente Logo**: formação e atuação. Campinas: Gráfica Central da UNICAMP, 1996. p.1-34.

VALENTE, José A.; VALENTE, Ann B. **Logo**: conceitos, aplicações e projetos. São Paulo: ITAUTEC Informática, 1988.

APÊNDICE A – Palavras reservadas da linguagem

A relação das palavras reservadas da linguagem é apresentada no quadro 34.

```
abs = IDENTIFICADOR : "abs"
algoritmo = IDENTIFICADOR : "algoritmo"
ate = IDENTIFICADOR : "ate"
ate_acento = IDENTIFICADOR : "até"
cadeia = IDENTIFICADOR : "cadeia"
caractere = IDENTIFICADOR : "caractere"
caso = IDENTIFICADOR : "caso"
de = IDENTIFICADOR : "de"
div = IDENTIFICADOR : "div"
e = IDENTIFICADOR : "e"
enquanto = IDENTIFICADOR : "enquanto"
entao = IDENTIFICADOR : "entao"
entao_acento = IDENTIFICADOR : "então"
escolha = IDENTIFICADOR : "escolha"
escreva = IDENTIFICADOR : "escreva"
faca = IDENTIFICADOR : "faca"
faca_acento = IDENTIFICADOR : "faça"
falso = IDENTIFICADOR : "f"
fim = IDENTIFICADOR : "fim"
fimenquanto = IDENTIFICADOR : "fimenquanto"
fimescolha = IDENTIFICADOR : "fimescolha"
fimpara = IDENTIFICADOR : "fimpara"
fimse = IDENTIFICADOR : "fimse"
inicio = IDENTIFICADOR : "inicio"
inicio_acento = IDENTIFICADOR : "início"
int = IDENTIFICADOR : "int"
inteiro = IDENTIFICADOR : "inteiro"
leia = IDENTIFICADOR : "leia"
limpar = IDENTIFICADOR : "limpartela"
logico = IDENTIFICADOR : "logico"
logico_acento = IDENTIFICADOR : "lógico"
matriz = IDENTIFICADOR : "matriz"
mod = IDENTIFICADOR : "mod"
ou = IDENTIFICADOR : "ou"
para = IDENTIFICADOR : "para"
real = IDENTIFICADOR : "real"
repita = IDENTIFICADOR : "repita"
se = IDENTIFICADOR : "se"
senao = IDENTIFICADOR : "senao"
senao_acento = IDENTIFICADOR : "senão"
tipo = IDENTIFICADOR : "tipo"
var = IDENTIFICADOR : "var"
verdadeiro = IDENTIFICADOR : "v"
```

Quadro 34 – Palavras reservadas

APÊNDICE B – Gramática com ações semânticas

A linguagem possui a seguinte estrutura:

a) um único bloco de comandos que inicia com **algoritmo** e termina com **fim** (esquema de tradução do quadro 36);
b) declaração de tipos, de uso exclusivo para declaração de matrizes unidimensionais (esquema de tradução do quadro 37);
c) declaração de variáveis (esquema de tradução do quadro 38);
d) comando de atribuição (esquema de tradução do quadro 39);
e) comando de entrada via teclado (esquema de tradução do quadro 40);
f) comando de saída (esquema de tradução do quadro 41);
g) comando de repetição enquanto-faça (esquema de tradução do quadro 42);
h) comando de repetição repita-até (esquema de tradução do quadro 43);
i) comando de repetição para-faça (esquema de tradução do quadro 44);
j) comando de seleção se-então (esquema de tradução do quadro 45);
k) comando de seleção escolha-caso (esquema de tradução do quadro 46);
l) comando para limpar a tela (esquema de tradução do quadro 47);
m) expressões aritméticas, lógicas e relacionais (esquema de tradução do quadro 48).

A gramática com as ações semânticas encontra-se no quadro 35.

```
<algoritmo> ::=      algoritmo IDENTIFICADOR #1 ";"
                            <declaracao_tipo>
                            <declaracao_variaveis>
                     <inicio> #2 <lista_comandos> fim "." #3 ;
<inicio> ::= inicio | inicio_acento ;

<declaracao_tipo> ::= ε |
#4 tipo <lista_identificadores> "=" matriz "["<faixa>"]" <tipo_primitivo> #5";" <tipos>;
<tipos> ::= ε |
   tipo <lista_identificadores> "=" matriz "["<faixa>"]" <tipo_primitivo> #5";" <tipos>;
<faixa> ::= CONSTANTE_INTEIRO #6 ".." CONSTANTE_INTEIRO #7 ;

<declaracao_variaveis> ::= #8 var <variaveis> | ε ;
<variaveis> ::= <lista_identificadores> ":" <tipo> ";" <variaveis_1> ;
<variaveis_1> ::= <variaveis> | ε ;
<tipo> ::= IDENTIFICADOR #9 | <tipo_primitivo>  ;
<tipo_primitivo> ::= inteiro #10 |
                     real #11 |
                     logico #12 | logico_acento #12 |
                     caractere #13 |
                     cadeia #14 ;

<lista_identificadores> ::= IDENTIFICADOR #15 <lista_identificadores_1> ;
<lista_identificadores_1> ::= "," IDENTIFICADOR #15 <lista_identificadores_1> | ε ;

<lista_comandos> ::= <comando> <lista_comandos_1>  ;
<lista_comandos_1> ::= <lista_comandos>  |  ε ;

<comando> ::= <atribuicao>|<entrada>|<saida>|<repeticao>|<selecao>|<limparTela> ;

<atribuicao> ::= IDENTIFICADOR <atribuicao_1> ;
<atribuicao_1> ::=   #16 ":=" <expressao> #101 #17 ";" |
                     #102 "[" <expressao> #103 "]" ":=" <expressao> #101 #18 ";" ;

<entrada> ::= leia "(" <lista_entrada> ")" ";" ;
<lista_entrada> ::= <variavel_entrada> <lista_entrada_1> ;
<lista_entrada_1> ::= "," <variavel_entrada> <lista_entrada_1> | ε ;
<variavel_entrada> ::= IDENTIFICADOR  <variavel_entrada_1>  ;
<variavel_entrada_1> ::= #102 "[" <expressao> #103 "]" #19 | #20 ε ;

<saida> ::= escreva "(" <lista_saida> ")" ";" ;
<lista_saida> ::= <item> #21 <lista_saida_1> ;
<lista_saida_1> ::= "," <item> #21 <lista_saida_1>  |  ε ;
<item> ::= <variavel_saida> |  <numero> |  CONSTANTE_LITERAL #22 ;
<variavel_saida> ::= IDENTIFICADOR <variavel_saida_1> ;
<variavel_saida_1> ::= #102 "[" <expressao> #103 "]" #23 | #24 ε ;

<repeticao> ::= <enquanto> | <repita> | <para> ;

<enquanto>::= enquanto #25 <expressao> #104 #26 <faca>
                     <lista_comandos>
              fimenquanto ";" #27;
<faca> ::= faca | faca_acento ;
<ate> ::= ate | ate_acento ;

<repita> ::=  repita #28 <lista_comandos> <ate> <expressao> #104 #29 ";" ;

<para> ::=    para IDENTIFICADOR #16
              de <inicio_para> #25 #30 <ate> #31 <expressao> #103 #32 #26
              <faca> <lista_comandos> fimpara #27 ";" ;
<inicio_para> ::= <expressao> #103 #33 | ε #34 ;

<selecao> ::= <se> | <escolha> ;

<se>::= se <expressao> #104 #35 <entao> <lista_comandos> <clausula_senao> fimse ";" #36;
<entao> ::= entao | entao_acento ;
<clausula_senao> ::= <senao> #37 <lista_comandos> |  ε ;
<senao> ::= senao | senao_acento ;

<escolha> ::= #38 escolha IDENTIFICADOR #16 #66
```

```
                    caso <opcao> #105 #39 #44 #35 ":" <lista_comandos> #40 <casos>
             <clausula_senao_escolha> fimescolha #41 ";" ;
<opcao> ::= CONSTANTE_INTEIRO #42 #56 | CONSTANTE_LITERAL #22 #62 ;
<casos> ::= caso <opcao>#105 #39 #44 #35 ":" <lista_comandos> #40 <casos> | ε ;
<clausula_senao_escolha> ::= <senao> <lista_comandos> | ε ;

<limparTela> ::= limpar ";"#43 ;

<expressao> ::=      <expressao_aritmetica_logica> <expressao_1> ;
<expressao_1> ::=    "=" <expressao_aritmetica_logica> #105 #44 #45 |
                     "<>" <expressao_aritmetica_logica> #105 #46 #45 |
                     "<" <expressao_aritmetica_logica> #105 #47 #45 |
                     ">" <expressao_aritmetica_logica> #105 #48 #45 |
                     "<=" <expressao_aritmetica_logica> #105 #49 #45 |
                     ">=" <expressao_aritmetica_logica> #105 #50 #45 | ε  ;
<expressao_aritmetica_logica> ::= <termo> <menor_prioridade> ;
<menor_prioridade> ::=      "+" <termo> #105 #51 <menor_prioridade> |
                            "-" <termo> #105 #52<menor_prioridade> |  ε ;
<termo> ::= <elemento> <maior_prioridade> ;
<maior_prioridade> ::=      "*" <elemento> #105  #53 <maior_prioridade> |
                            "/"#54 #105 <elemento> #105  #55 <maior_prioridade> |
                            div #56 #105 <elemento> #105 #57 <maior_prioridade> |
                            e #58 #105 <elemento> #105 #59 <maior_prioridade> |
                            ou #58 #105 <elemento> #105 #60<maior_prioridade> |
                            mod#56 #105 <elemento> #105 #61 <maior_prioridade> | ε ;
<elemento> ::=       <variavel> |
                     <numero> |
                     CONSTANTE_LITERAL #22 #62 |
                     <constante_logica> #58 |
                     "(" <expressao>")" |
                     <sinal> <elemento> #63 |
                     abs "("<expressao>")" #64 |
                     int "("<expressao>")" #65  ;

<variavel> ::= IDENTIFICADOR <variavel_1> ;
<variavel_1> ::= #102 #66 "[" <expressao> #103 "]" #23 | #66 #24 ε ;

<constante_logica> ::= verdadeiro #67 |  falso #68 ;

<numero> ::= CONSTANTE_INTEIRO #42 #56 | CONSTANTE_REAL #69 #54 ;

<sinal> ::= "+"#70 | "-"#70 ;
```

Quadro 35 – Gramática com ações semânticas

SINTAXE	**EXEMPLO**
<algoritmo> → algoritmo IDENTIFICADOR **#1** ";" <declaracao_tipo> <declaracao_variaveis> <inicio>**#2** <lista_comandos> fim "." **#3**	ALGORITMO primeiro_algoritmo; INÍCIO ESCREVA ('oi mundo!'); FIM.
ação#1: Adicionar o nome do algoritmo na tabela de símbolos. **ação#2:** Informar a linha onde iniciam os comandos do algoritmo. **ação#3:** Gerar a instrução STP (final do algoritmo). Informar se alguma variável não foi utilizada ou não foi inicializada.	

Quadro 36 – Esquema de tradução do algoritmo

SINTAXE	EXEMPLO

<declaracao_tipo> ::=
  #4 tipo <lista_identificadores> "=" matriz
  "["<faixa>"]" <tipo_primitivo> **#5** ";" <tipos> | ε ;
 	
<tipos> ::=
  tipo <lista_identificadores> "=" matriz
  "["<faixa>"]" <tipo_primitivo> **#5** ";" <tipos> | ε ;
 	
<faixa> ::=
  CONSTANTE_INTEIRO **#6** ".." CONSTANTE_INTEIRO **#7** ;
	ALGORITMO com_tipo;
	TIPO
	  vetor = MATRIZ [1..10] INTEIRO;
	VAR
	  valor : vetor;
	  média : REAL;
	início
	  LEIA (valor[1], valor[2]);
	  média:= (valor[1]+valor[2])/2;
	  ESCREVA (média);
	FIM.

ação#4: Informar que o contexto é de declaração de tipos. **ação#5:** Adicionar na tabela de símbolos qual o tipo dos elementos da matriz. **ação#6:** Adicionar na tabela de símbolos qual o limite inferior da matriz, isto é, onde inicia a faixa. **ação#7:** Verificar se o limite superior da matriz é maior que o limite inferior. Em caso afirmativo, adicionar na tabela de símbolos qual o limite superior da matriz.	

Quadro 37 – Esquema de tradução da declaração de tipos

SINTAXE	EXEMPLO

<declaracao_variaveis> ::= **#8** var <variaveis> | ε
 	
<variaveis> ::=
  <lista_identificadores>":" <tipo> ";" <variaveis_l>
<variaveis_1> ::= <variaveis> | ε
 	
<tipo> ::= IDENTIFICADOR **#9** | <tipo_primitivo>
<tipo_primitivo> ::= inteiro **#10** |
  real **#11** |
  logico **#12** | logico_acento **#12** |
  caractere **#13** |
  cadeia **#14**
 	
<lista_identificadores> ::=
  IDENTIFICADOR **#15** <lista_identificadores_1>
<lista_identificadores_1> ::=
  "," IDENTIFICADOR **#15** <lista_identificadores_1> | ε
	ALGORITMO com_variáveis;
	VAR
	  valorA, valorB : INTEIRO;
	  média : REAL ;
	INÍCIO
	  LEIA (valorA, valorB);
	  média := (valorA+valorB)/2;
	  ESCREVA (média);
	FIM.

ação#8: Informar que o contexto é de declaração de variáveis. **ação#9:** Verificar se o tipo indicado foi declarado (existe na tabela de símbolos). Em caso afirmativo, recuperar o tipo (categoria) e calcular o tamanho da matriz; alterar na tabela de símbolos o tipo (categoria) e o endereço (atributo) das últimas variáveis declaradas (são do tipo matriz); gerar a instrução para alocar memória para as variáveis declaradas; atualizar o contador de variáveis. **ação#10:** Se o contexto for de declaração de variáveis, alterar na tabela de símbolos o tipo (categoria) das últimas variáveis declaradas (são do tipo inteiro); gerar instrução ALI (alocar memória para as variáveis do tipo inteiro). **ação#11:** Se o contexto for de declaração de variáveis, alterar na tabela de símbolos o tipo (categoria) das últimas variáveis declaradas (são do tipo real); gerar instrução ALR (alocar memória para as variáveis do tipo real). **ação#12:** Se o contexto for de declaração de variáveis, alterar na tabela de símbolos o tipo (categoria) das últimas variáveis declaradas (são do tipo lógico); gerar instrução ALB (alocar memória para as variáveis do tipo lógico). **ação#13:** Se o contexto for de declaração de variáveis, alterar na tabela de símbolos o tipo (categoria) das últimas variáveis declaradas (são do tipo caractere); gerar instrução ALC (alocar memória para as variáveis do tipo caractere). **ação#14:** Se o contexto for de declaração de variáveis, alterar na tabela de símbolos o tipo (categoria) das últimas variáveis declaradas (são do tipo cadeia); gerar a instrução ALS (alocar memória para as variáveis do tipo cadeia). **ação#15:** Se o contexto for declaração de tipos, verificar se o identificador já foi declarado (existe na tabela de símbolos). Em caso negativo, adicionar o identificador na tabela de símbolos (é o nome de um novo tipo). Se o contexto for de declaração de variáveis, verificar se o identificador já foi declarado (existe na tabela de símbolos). Em caso negativo, incrementar o contador de variáveis e adicionar o identificador na tabela de símbolos (é o nome de uma variável).	

Quadro 38 – Esquema de tradução da declaração de variáveis

SINTAXE	EXEMPLO

<atribuicao> ::= IDENTIFICADOR <atribuicao_1>
<atribuicao_1> ::=
  #16 ":=" <expressao> **#101 #17** ";" |
  #102 "[" <expressao> **#103** "]"
  ":=" <expressao> **#101 #18** ";"
	ALGORITMO com_atribuição;
	VAR
	  valorA, valorB : INTEIRO;
	  média : REAL ;
	INÍCIO
	  ESCREVA ('digite valores:');
	  LEIA (valorA, valorB);
	  média:= (valorA+valorB)/2;
	  ESCREVA ('a média é:', média);
	FIM.

ação#16: Verificar se o identificador foi declarado (existe na tabela de símbolos) e se é variável de tipo primitivo. Se for, armazenar o identificador para uso em outra ação semântica. **ação#17:** Alterar na tabela de símbolos as propriedades "usado" e "inicializado" do identificador (reconhecido pela ação #16) para verdadeiro. Gerar instrução STR (armazenar valor). **ação#18:** Alterar na tabela de símbolos as propriedades "usado" e "inicializado" do identificador (reconhecido pela ação #102) para verdadeiro. Gerar a instrução STA (armazenar valor em uma variável do tipo matriz). **ação#101:** Verificar se o tipo da expressão é compatível com o tipo do identificador. **ação#102:** Verificar se o identificador foi declarado (existe na tabela de símbolos) e se é variável do tipo matriz. Se for, armazenar o identificador para uso em outra ação semântica. **ação#103:** Verificar se a expressão é do tipo inteiro. Se não for, informar incompatibilidade de tipos (o índice de uma matriz só pode ser do tipo inteiro).	

Quadro 39 – Esquema de tradução do comando de atribuição

SINTAXE	EXEMPLO

<entrada> ::= leia "(" <lista_entrada> ")" ";"
 	
<lista_entrada>::=<variavel_entrada><lista_entrada_1>
<lista_entrada_1> ::=
  "," <variavel_entrada> <lista_entrada_1> | ε
 	
<variavel_entrada> ::=
  IDENTIFICADOR <variavel_entrada_1>
<variavel_entrada_1> ::=
  #102 "[" <expressao> **#103** "]" **#19** | **#20** ε ;
	ALGORITMO entrada_de_dados;
	VAR
	  nome01, nome02 : CADEIA;
	INÍCIO
	  ESCREVA ('digite dois nomes');
	  LEIA (nome01, nome02);
	FIM.

ação#19: Gerar instrução REA (entrada de dados) com parâmetro igual ao tipo (categoria) do identificador (reconhecido pela ação #102). Gerar a instrução STA (armazenar o valor em uma variável do tipo matriz). Alterar na tabela de símbolos as propriedades "usado" e "inicializado" do identificador (reconhecido pela ação #102) para verdadeiro. **ação#20:** Verificar se o identificador foi declarado (existe na tabela de símbolos) e se é variável de tipo primitivo. Se for, gerar instrução REA (entrada de dados); gerar instrução STR (armazenar valor); alterar na tabela de símbolos as propriedades "usado" e "inicializado" do identificador para verdadeiro.	

Quadro 40 – Esquema de tradução do comando de entrada

SINTAXE	EXEMPLO
<saida> ::= escreva "(" <lista_saida> ")" ";" <lista_saida> ::= <item> **#21** <lista_saida_1> <lista_saida_1> ::= "," <item> **#21** <lista_saida_1> | ε <item> ::= <variavel_saida>|<numero>|CONSTANTE_LITERAL **#22** <variavel_saida>::= IDENTIFICADOR <variavel_saida_1> <variavel_saida_1> ::= **#102** "[" <expressao> **#103** "]" **#23** | **#24** ε	ALGORITMO com_saída_de_dados; INÍCIO ESCREVA ('oi mundo!'); FIM .
ação#21: Gerar instrução WRT (escrever valor). **ação#22:** Gerar instrução LDS (carregar uma constante literal). **ação#23:** Gerar instrução LDA (carregar o valor de uma variável do tipo matriz). Alterar na tabela de símbolos a propriedade "usado" do identificador para verdadeiro. **ação#24:** Verificar se o identificador foi declarado (existe na tabela de símbolos) e se é variável de tipo primitivo. Em caso afirmativo, gerar instrução LDV (carregar o valor da variável) e alterar na tabela de símbolos a propriedade "usado" do identificador para verdadeiro.	

Quadro 41 – Esquema de tradução do comando de saída

SINTAXE	EXEMPLO
<enquanto>::= enquanto **#25** <expressao> **#104** **#26** <faca> <lista_comandos> fimenquanto ";" **#27**	ALGORITMO com_ enquanto; VAR Gasto, gastos : INTEIRO; INÍCIO Gastos := 0; ESCREVA ('escreva o gasto:'); LEIA (gasto); ENQUANTO (gasto <> 0) FAÇA gastos := gastos + gasto; LEIA (gasto); FIMENQUANTO; ESCREVA (gastos); FIM.
ação#25: Empilhar o endereço (ponteiro) da próxima instrução a ser gerada. **ação#26:** Gerar a instrução JMF (desvio). Empilhar o endereço da instrução JMF. **ação#27:** Atualizar o parâmetro da instrução de desvio (JMF) gerada na ação #26 com o endereço (ponteiro+1) da próxima instrução a ser gerada. Desempilhar o endereço da instrução de desvio (JMF). Gerar a instrução JMP (desvio) com parâmetro igual ao endereço do topo da pilha. Desempilhar o endereço empilhado na ação #25. **ação#104:** Verificar se a expressão é do tipo lógico. Se não for, informar incompatibilidade de tipos (uma expressão em um comando de repetição só pode ser do tipo lógico).	

Quadro 42 – Esquema de tradução do comando enquanto-faça

SINTAXE	EXEMPLO

<repita> ::=
repita **#28**
<lista_comandos>
<ate> <expressao> **#104** **#29** ";"
	ALGORITMO com_repita;
	VAR
	gasto, gastos : INTEIRO;
	INÍCIO
	gastos := 0;
	REPITA
	ESCREVA ('escreva o gasto:');
	LEIA (gasto);
	gastos := gastos + gasto;
	ATÉ gasto = 0;
	ESCREVA (gastos);
	FIM.

ação#28: Empilhar o endereço (ponteiro) da próxima instrução a ser gerada. **ação#29:** Gerar a instrução JMF (desvio). Desempilhar o endereço empilhado na ação #28.	

Quadro 43 – Esquema de tradução do comando repita-até

SINTAXE	EXEMPLO

<para> ::=
  para IDENTIFICADOR **#16**
  de <inicio_para> **#25 #30**
  <ate> **#31** <expressao> **#103 #32 #26** <faca>
  <lista_comandos>
  fimpara **#27** ";"
 	
<inicio_para> ::= <expressao> **#103 #33** | ε **#34** ;
	ALGORITMO com_para;
	TIPO
	  Vetor = MATRIZ [1..10] INTEIRO;
	VAR
	  Contador : INTEIRO;
	  valores : vetor;
	INÍCIO
	  PARA contador DE 1 ATÉ 10 FAÇA
	  valores[contador]:=
	  contador+10;
	  FIMPARA;
	FIM.

ação#30: Gerar instrução LDV (carregar o valor da variável - identificador reconhecido pela ação #16). Gerar instrução LDI (carregar a constante 1). Gerar instrução ADD (soma). Gerar instrução STR (armazenar valor). **ação#31:** Gerar instrução LDV (carregar o valor da variável). **ação#32:** Gerar instrução SME (menor ou igual). **ação#33:** Alterar na tabela de símbolos a propriedade "inicializado" do identificador (reconhecido pela ação #16) para verdadeiro. Gerar instrução LDI (carregar a constante 1). Gerar instrução SUB (subtração). Gerar instrução STR (armazenar valor). **ação#34:** Alterar na tabela de símbolos a propriedade "inicializado" do identificador (reconhecido pela ação #16) para verdadeiro. Gerar instrução LDI (carregar a constante 1). Gerar instrução STR (armazenar valor).	

Quadro 44 – Esquema de tradução do comando para-faça

SINTAXE	EXEMPLO

<se> ::=
  se <expressao> **#104 #35** <entao>
  <lista_comandos>
  <clausula_senao>
  fimse ";" **#36**
 	
<clausula_senao> ::=
  <senao> **#37** <lista_comandos> | ε
	ALGORITMO com_se;
	VAR
	  nota : INTEIRO;
	INÍCIO
	  ESCREVA ('digite a nota:');
	  LEIA (nota);
	  SE nota > 6 ENTÃO
	  ESCREVA ('Aprovado');
	  SENÃO
	  ESCREVA('Reprovado');
	  FIMSE;
	FIM.

ação#35: Gerar a instrução JMF (desvio). Empilhar o endereço da instrução JMF. **ação#36:** Atualizar o parâmetro da instrução de desvio (JMF ou JMP) com o endereço (ponteiro) da próxima instrução a ser gerada. Desempilhar o endereço da instrução de desvio. **ação#37:** Atualizar o parâmetro da instrução de desvio (JMF) gerada na ação #35 com o endereço (ponteiro+1) da próxima instrução a ser gerada. Desempilhar o endereço da instrução de desvio (JMF). Gerar a instrução JMP (desvio). Empilhar o endereço da instrução JMP.	

Quadro 45 – Esquema de tradução do comando se-senão

SINTAXE	EXEMPLO

<escolha> ::=
  #38 escolha IDENTIFICADOR **#16 #66**
  caso <opcao> **#105 #39 #44 #35** ":"
  <lista_comandos> **#40** <casos>
  <clausula_senao_escolha>
  fimescolha **#41** ";"
<opcao> ::= CONSTANTE_INTEIRO **#42 #56** |
  CONSTANTE_LITERAL **#22 #62**
<casos> ::=
  caso <opcao>**#105 #39 #44 #35** ":"
  <lista_comandos> **#40** <casos> | ε
 	
<clausula_senao_escolha> ::=
  <senao> <lista_comandos> | ε ;
	ALGORITMO com_escolha;
	VAR
	  nota : INTEIRO;
	INÍCIO
	  ESCREVA ('digite a nota:');
	  LEIA (nota);
	  ESCOLHA nota
	  Caso 7: ESCREVA('Aprovado');
	  Caso 8: ESCREVA('Aprovado');
	  Caso 9: ESCREVA('Aprovado');
	  Caso 10: ESCREVA('Parabéns');
	  SENÃO
	  ESCREVA('Reprovado');
	  FIMESCOLHA;
	FIM.

ação#38: Incrementar o controle de desvio. **ação#39:**Gerar instrução LDV (carregar o valor da variável - identificador reconhecido pela ação #16). **ação#40:**Atualizar o parâmetro da instrução de desvio (JMF) gerada na ação #35 com o endereço (ponteiro+1) da próxima instrução a ser gerada. Desempilhar o endereço da instrução de desvio (JMF). Gerar a instrução JMP (desvio). Empilhar o endereço da instrução JMP. **ação#41:**Desempilhar a quantidade de desvios pendentes (controle de desvio). Decrementar o controle de desvio. Alterar na tabela de símbolos a propriedade “usado” do identificador (reconhecido pela ação #16) para verdadeiro. **ação#42:**Gerar a instrução LDI (carregar constante inteira). **ação#44:**Gera a instrução EQL (igual). **ação#56:**Indicar que é esperado um valor do tipo inteiro. **ação#62:**Indicar que é esperado um valor do tipo cadeia. **ação#66:**Indicar que é esperado um valor de tipo compatível com o tipo do identificador. **ação#105:**Verificar se o tipo da opção é compatível com o tipo do identificador.	

Quadro 46 – Esquema de tradução do comando escolha-caso

SINTAXE	EXEMPLO
<limparTela> ::= limpar ";" **#43**	ALGORITMO com_limparTela; INÍCIO limparTela; FIM.
ação#43: Gerar instrução CLR (limpar a tela).	

Quadro 47 – Esquema de tradução do comando limparTela

SINTAXE

```
<expressao> ::= <expressao_aritmetica_logica> <expressao_1> ;
<expressao_1> ::=    "=" <expressao_aritmetica_logica> #105 #44 #45 |
                     "<>" <expressao_aritmetica_logica> #105 #46 #45 |
                     "<" <expressao_aritmetica_logica> #105 #47 #45 |
                     ">"<expressao_aritmetica_logica> #105 #48 #45 |
                     "<="<expressao_aritmetica_logica> #105 #49 #45 |
                     ">="<expressao_aritmetica_logica> #105 #50 #45 | ε
<expressao_aritmetica_logica> ::= <termo> <menor_prioridade>
<menor_prioridade>::=       "+"<termo> #105 #51 <menor_prioridade> |
                            "-"<termo> #105 #52 <menor_prioridade> |  ε
<termo> ::= <elemento> <maior_prioridade>
<maior_prioridade> ::=      "*" <elemento> #105 #53 <maior_prioridade> |
                            "/"#54 #105 <elemento> #105 #55<maior_prioridade> |
                            div #56 #105 <elemento> #105 #57 <maior_prioridade> |
                            e #58 #105 <elemento> #105 #59 <maior_prioridade> |
                            ou #58 105 <elemento> #105 #60 <maior_prioridade> |
                            mod #56 #105 <elemento> #105 #61 <maior_prioridade> |ε
<elemento> ::=       <variavel> |
                     <numero> |
                     CONSTANTE_LITERAL #22 #62 |
                     <constante_logica> #58 |
                     "(" <expressao>")" |
                     <sinal> <elemento> #63 |
                     abs "("<expressao>")" #64 |
                     int "("<expressao>")" #65
<variavel> ::= IDENTIFICADOR <variavel_1>
<variavel_1> ::= #102 #66 "[" <expressao> #103 "]" #23 | #66 #24 ε
<constante_logica> ::= verdadeiro #67 |  falso #68
<numero> ::= CONSTANTE_INTEIRO #42 #56 | CONSTANTE_REAL #69 #54
<sinal> ::= "+" #70 | "-" #70
```

ação#45: Indicar que é esperado um valor do tipo lógico.
ação#46: Gerar instrução DIF (diferente).
ação#47: Gerar instrução SMR (menor que).
ação#48: Gerar instrução BGR (maior que).
ação#49: Gerar instrução SME (menor ou igual).
ação#50: Gerar instrução BGE (operação >=).
ação#51: Verificar se a expressão é do tipo inteiro ou real. Se for, gerar instrução ADD (soma).
ação#52: Verificar se a expressão é do tipo inteiro ou real. Se for, gerar instrução SUB (subtração).
ação#53: Verificar se a expressão é do tipo inteiro ou real. Se for, gerar instrução MUL (multiplicação).
ação#54: Indicar que é esperado um valor do tipo real.
ação#55: Verificar se a expressão é do tipo inteiro ou real. Se for, gerar instrução DIVR (divisão de números reais).
ação#57: Verificar se a expressão é do tipo inteiro. Se for, gerar instrução DIVI (divisão de números inteiros).
ação#58: Indicar que é esperado um valor do tipo lógico.
ação#59: Gerar instrução AND (operação E).
ação#60: Gerar instrução OR (operação OU).
ação#61: Verificar se a expressão é do tipo inteiro. Se for, gerar instrução MOD (resto da divisão de números inteiros).
ação#63: Se sinal reconhecido na ação #70 for igual a **–**, gerar instrução LDI (carregar a constante –1); gerar instrução MUL (multiplicação); atribuir **+** para sinal.
ação#64: Verificar se a expressão é do tipo inteiro ou real. Se for, gerar instrução ABS (função ABS).
ação#65: Verificar se a expressão é do tipo inteiro ou real. Se for, gerar instrução INT (função INT); indicar que é esperado um valor do tipo inteiro.
ação#67: Gerar instrução LDB (carregar a constante lógica verdadeiro).
ação#68: Gerar instrução LDB (carregar a constante lógica falso).
ação#69: Gerar a instrução LDR (carregar constante real).
ação#70: Armazenar o sinal reconhecido.

Quadro 48 – Esquema de tradução das expressões

APÊNDICE C – Instruções da linguagem intermediária

A relação das instruções da linguagem intermediária é apresentada no quadro 49.

CÓDIGO	PARÂMETRO	FUNCIONAMENTO	RESTRIÇÕES
ABS	0	Retirar o topo da memória; calcular o valor absoluto e colocar o resultado no topo da memória.	Somente para valores numéricos.
ADD	0	Retirar os dois últimos valores da memória; somar os dois valores e colocar o resultado no topo da memória.	Somente para valores numéricos.
ALB	número de posições	Alocar na memória o número de posições indicado no parâmetro.	Para valores do tipo lógico.
ALC	número de posições	Alocar na memória o número de posições indicado no parâmetro.	Para valores do tipo caractere.
ALI	número de posições	Alocar na memória o número de posições indicado no parâmetro.	Para valores do tipo inteiro.
ALR	número de posições	Alocar na memória o número de posições indicado no parâmetro.	Para valores do tipo real.
ALS	número de posições	Alocar na memória o número de posições indicado no parâmetro.	Para valores do tipo cadeia.
AMB	tamanho da matriz	Alocar na memória o número de posições indicado no parâmetro, referente ao tamanho da matriz.	Para matriz do tipo lógico.
AMC	tamanho da matriz	Alocar na memória o número de posições indicado no parâmetro, referente ao tamanho da matriz.	Para matriz do tipo caractere.
AMI	tamanho da matriz	Alocar na memória o número de posições indicado no parâmetro, referente ao tamanho da matriz.	Para matriz do tipo inteiro.
AMR	tamanho da matriz	Alocar na memória o número de posições indicado no parâmetro, referente ao tamanho da matriz.	Para matriz do tipo real.
AMS	tamanho da matriz	Alocar na memória o número de posições indicado no parâmetro, referente ao tamanho da matriz.	Para matriz do tipo cadeia.
AND	0	Retirar os dois últimos valores da memória; efetuar a operação lógica E e colocar o resultado lógico no topo da memória.	
BGE	0	Retirar os dois últimos valores da memória; verificar se o penúltimo valor é maior ou igual ao último e colocar o resultado lógico no topo da memória.	
BGR	0	Retirar os dois últimos valores da memória; verificar se o penúltimo valor é maior que o último e colocar o resultado lógico no topo da memória.	
CLR	0	Limpar a tela de entrada de dados.	
DIF	0	Retirar os dois últimos valores da memória; verificar se são diferentes e colocar o resultado lógico no topo da memória.	
DIVI	0	Retirar os dois últimos valores da memória; verificar se o topo é diferente de zero; dividir o penúltimo valor pelo último e colocar o resultado no topo memória.	Somente para valores inteiros.
DIVR	0	Retirar os dois últimos valores da memória; verificar se o topo é diferente de zero; dividir o penúltimo	Somente para valores numéricos.

CÓDIGO	PARÂMETRO	FUNCIONAMENTO	RESTRIÇÕES
		valor pelo último e colocar o resultado no topo da memória.	
EQL	0	Retirar os dois últimos valores da memória; verificar se são iguais e colocar o resultado lógico no topo da memória.	
INT	0	Retirar o topo da memória; calcular o valor inteiro do número e colocar o resultado no topo da memória.	Somente para valores numéricos.
JMF	próxima instrução	Verificar se o topo da memória é igual a falso; se for, definir que a próxima instrução a ser executada é aquela cujo endereço foi indicado pelo parâmetro.	
JMP	próxima instrução	Definir que a próxima instrução a ser executada é aquela cujo endereço foi indicado pelo parâmetro.	
JMT	próxima instrução	Verificar se o topo da memória é igual a verdadeiro; se for, definir que a próxima instrução a ser executada é aquela cujo endereço foi indicado pelo parâmetro.	
LDV	endereço	Alocar uma posição na memória e colocar o valor que estiver na posição de memória indica pelo parâmetro na posição alocada.	Exceto matrizes.
LDA	posição	Calcular o tamanho da matriz; calcular o endereço desta posição da matriz na memória; verificar se esta posição pertence à matriz; alocar uma posição da memória; carregar o valor que estiver na posição de memória calculada; colocar o valor carregado na posição alocada.	Somente para matrizes.
LDB	constante	Alocar uma posição de memória e colocar o valor passado por parâmetro.	Somente para constante lógica.
LDI	constante	Alocar uma posição de memória e colocar o valor passado por parâmetro.	Somente para constante inteira.
LDR	constante	Alocar uma posição de memória e colocar o valor passado por parâmetro.	Somente para constante real.
LDS	constante	Alocar uma posição de memória e colocar o valor passado por parâmetro.	Somente para constante do tipo cadeia ou caractere.
MOD	0	Retirar os dois últimos valores da memória; efetuar a operação MOD que retorna o resto da divisão e colocar o resultado no topo da memória.	Somente para valores inteiros.
MUL	0	Retirar os dois últimos valores da memória; multiplicar os dois valores e colocar o resultado no topo da memória.	Somente para valores numéricos.
OR	0	Retirar os dois últimos valores da memória; efetuar a operação lógica OU e colocar o resultado lógico no topo da memória.	
REA	endereço	Habilitar a área de entrada de dados; quando o usuário digitar <enter>, verificar se o valor pode ser armazenado na variável; alocar uma posição de memória e colocar o valor nesta posição.	
SME	0	Retirar os dois últimos valores da memória; verificar se o penúltimo valor é menor ou igual ao último e colocar o resultado lógico no topo da	

CÓDIGO	PARÂMETRO	FUNCIONAMENTO	RESTRIÇÕES
		memória.	
SMR	0	Retirar os dois últimos valores da memória; verificar se o penúltimo valor é menor que último e colocar o resultado lógico no topo da memória.	
STA	posição	Calcular o tamanho da matriz; calcular o endereço desta posição da matriz na memória; verificar se esta posição pertence à matriz; colocar o valor do topo da memória na posição calculada; desalocar as duas últimas posições da memória.	Somente para matrizes.
STR	endereço	Colocar o valor do topo da memória na posição indicada pelo parâmetro.	Exceto para matrizes.
STP	0	Informar o final da execução.	
SUB	0	Retirar os dois últimos valores da memória; subtrair o penúltimo valor pelo último valor e colocar o resultado no topo da memória.	Somente para valores numéricos.
WRT	constante	Escrever o valor do topo da memória na tela, desalocar a última posição da memória.	

Quadro 49 – Instruções do código intermediário

APÊNDICE D – Questionários aplicados

Nos quadros 50 e 51 são apresentados os questionários respondidos pelos alunos quando da avaliação da ferramenta.

CONCEITO	5	4	3	2	1
1. No início da aula, quando você ainda não conhecia a ferramenta, foi fácil usá-la?					
	(5 – muito fácil; 4 – fácil; 3 – razoável; 2 – difícil; 1 – muito difícil)				
2. No final da aula, como você classifica a facilidade de uso da ferramenta?					
	(5 – muito fácil; 4 – fácil; 3 – razoável; 2 – difícil; 1 – muito difícil)				
3. A ferramenta responde de acordo com o esperado?					
	(5-sempre; 4-quase sempre; 3-algumas vezes; 2-quase nunca; 1- nunca)				
4. Os ícones e atalhos são condizentes com as funções associadas?					
	(5-sempre; 4-quase sempre; 3-algumas vezes; 2-quase nunca; 1- nunca)				
5. Como você classifica a interface da ferramenta?					
	(5-muito boa; 4-boa; 3-razoável; 2-ruim; 1-não consegui utilizar)				
6. As mensagens de erro da ferramenta levaram você a descobrir seu erro?					
	(5-sempre; 4-quase sempre; 3-algumas vezes; 2-quase nunca; 1- nunca)				
7. Como você classifica a ajuda sobre Portugol da ferramenta?					
	(5-muito boa; 4-boa; 3-razoável; 2-ruim; 1-não utilizei)				
8. A ferramenta auxiliou no aprendizado da disciplina?					
	(5-plenamente; 4-adequadamente;3-razoavelmente; 2-insuficientemente; 1 – não auxiliou)				
9. Você acha que o uso de uma ferramenta auxilia no aprendizado da disciplina					
	(5-Sim, está ferramenta ajuda; 4-Sim, mas talvez outra ferramenta fosse melhor; 3-Não, somente no papel seria suficiente)				
10. Que nota você daria para a ferramenta?					
CRÍTICAS /SUGESTÕES					

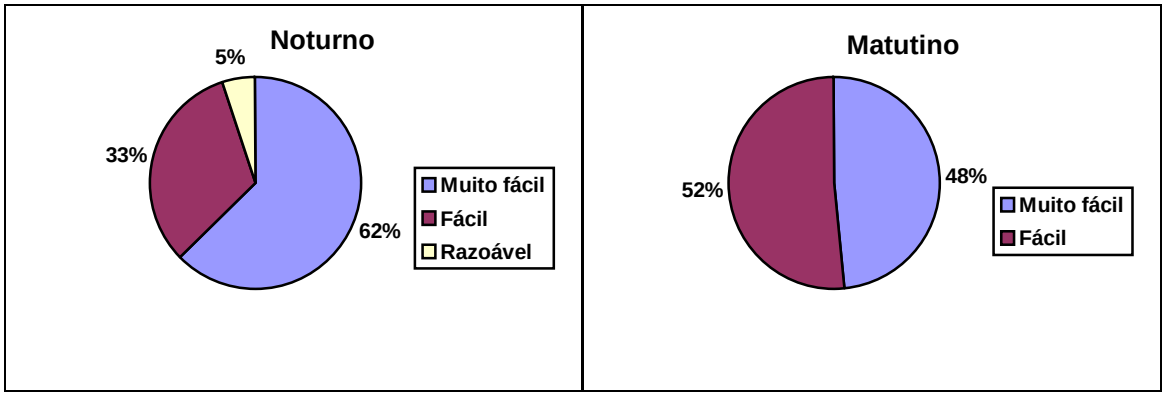
Quadro 50 – Questionário inicial

CONCEITO	5	4	3	2	1
1. Com que freqüência você utilizou a ferramenta?					
	(5 – diariamente; 4 – semanalmente; 3 – mensalmente; 2 – somente em sala de aula; 1 – não utilizei)				
2. No final da aula, como você classifica a facilidade de uso da ferramenta?					
	(5 – muito fácil; 4 – fácil; 3 – razoável; 2 – difícil; 1 – muito difícil)				
3. A ferramenta responde de acordo com o esperado?					
	(5-sempre; 4-quase sempre; 3-algumas vezes; 2-quase nunca; 1- nunca)				
4. Os ícones e atalhos são condizentes com as funções associadas?					
	(5-sempre; 4-quase sempre; 3-algumas vezes; 2-quase nunca; 1- nunca)				
5. Como você classifica a interface da ferramenta?					
	(5-muito boa; 4-boa; 3-razoável; 2-ruim; 1-não consegui utilizar)				
6. As mensagens de erro da ferramenta levaram você a descobrir seu erro?					
	(5-sempre; 4-quase sempre; 3-algumas vezes; 2-quase nunca; 1- nunca)				
7. Como você classifica a ajuda sobre Portugol da ferramenta?					
	(5-muito boa; 4-boa; 3-razoável; 2-ruim; 1-não utilizei)				
8. A ferramenta auxiliou no aprendizado da disciplina?					
	(5-plenamente; 4-adequadamente;3-razoavelmente; 2-insuficientemente; 1 – não auxiliou)				
9. Você acha que o uso de uma ferramenta auxilia no aprendizado da disciplina					
	(5-Sim, está ferramenta ajuda; 4-Sim, mas talvez outra ferramenta fosse melhor; 3-Não, somente no papel seria suficiente)				
10. Que nota você daria para a ferramenta?					
CRÍTICAS /SUGESTÕES					

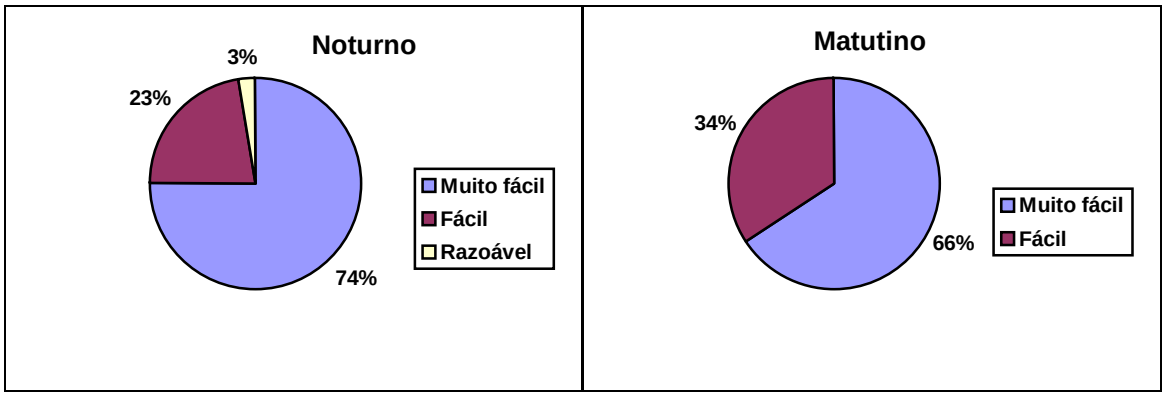
Quadro 51 – Questionário final

APÊNDICE E – Gráficos da avaliação dos alunos

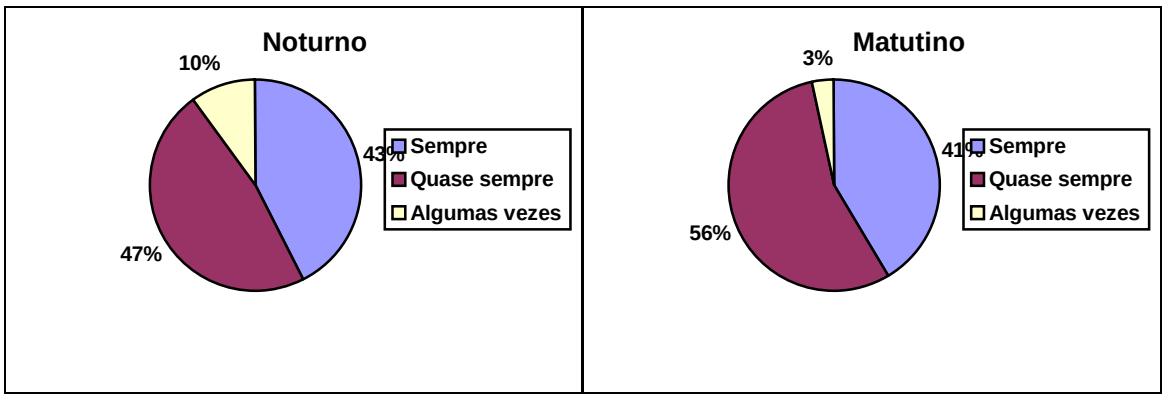
Nos quadros 52 até 61 encontram-se os gráficos com o resultado da avaliação inicial da ferramenta feita pelos alunos da disciplina de Introdução à Programação (turmas dos turnos matutino e do noturno).



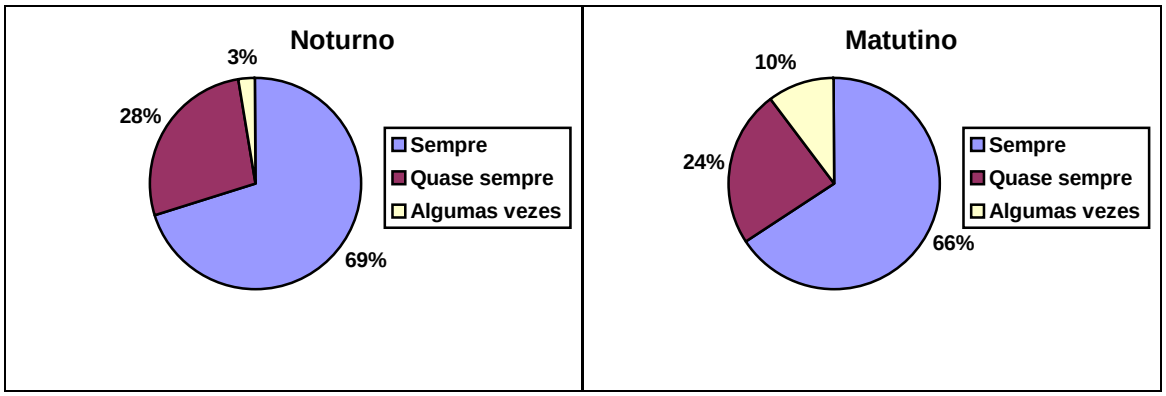
Quadro 52 – No início da aula, quando você ainda não conhecia a ferramenta, foi fácil usá-la?



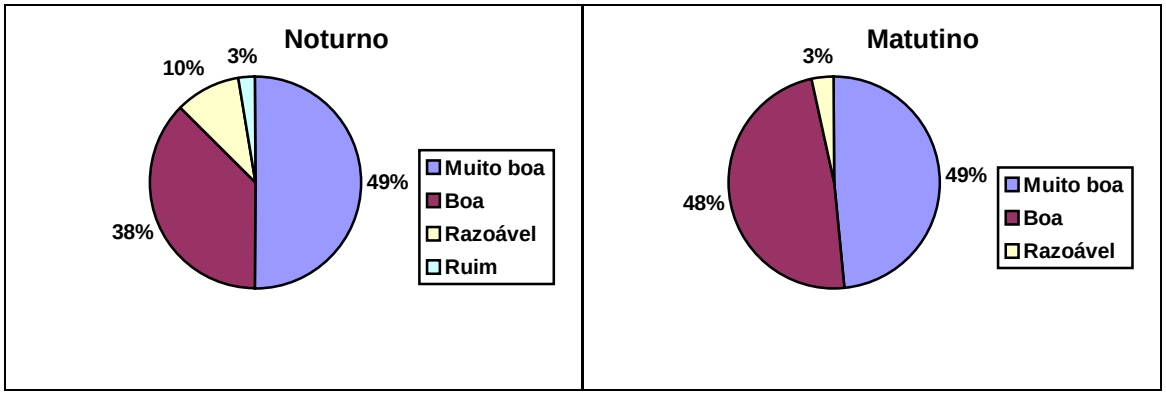
Quadro 53 – No final da aula, como você classifica a facilidade de uso da ferramenta?



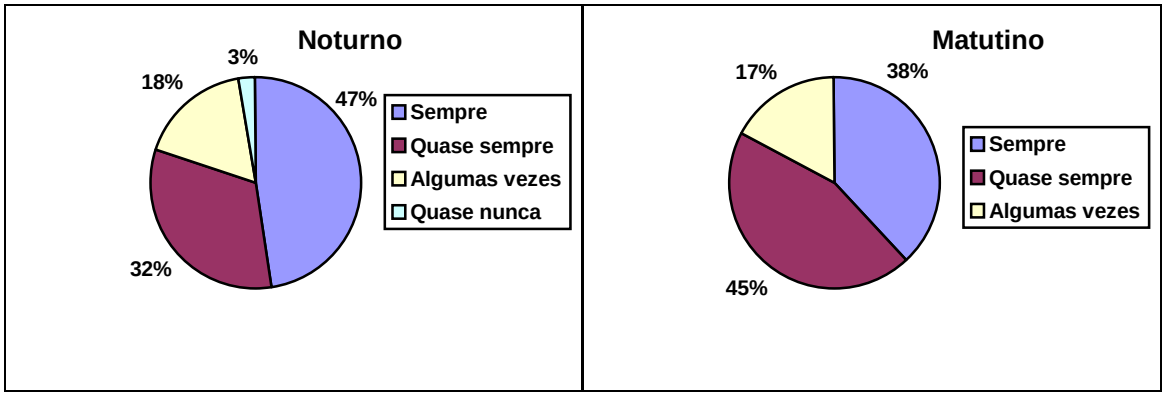
Quadro 54 – A ferramenta responde de acordo com o esperado?



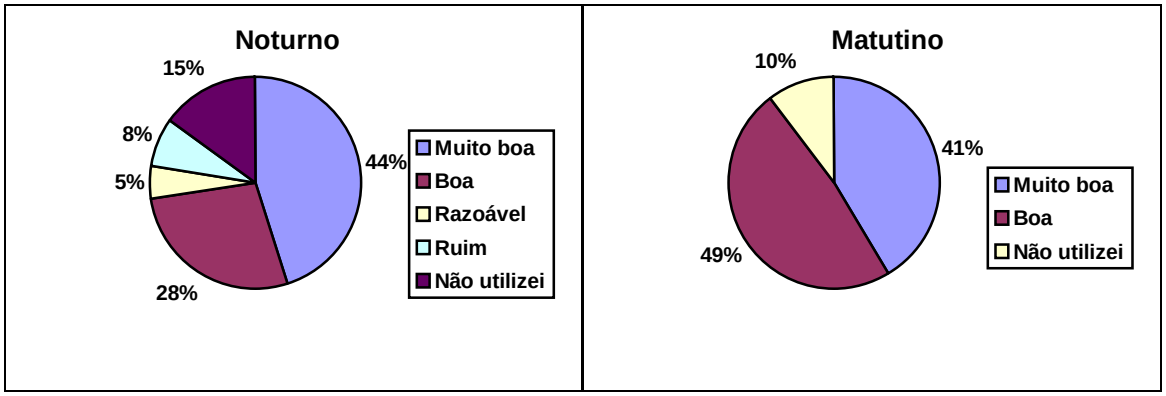
Quadro 55 – Os ícones e atalhos são condizentes com as funções associadas?



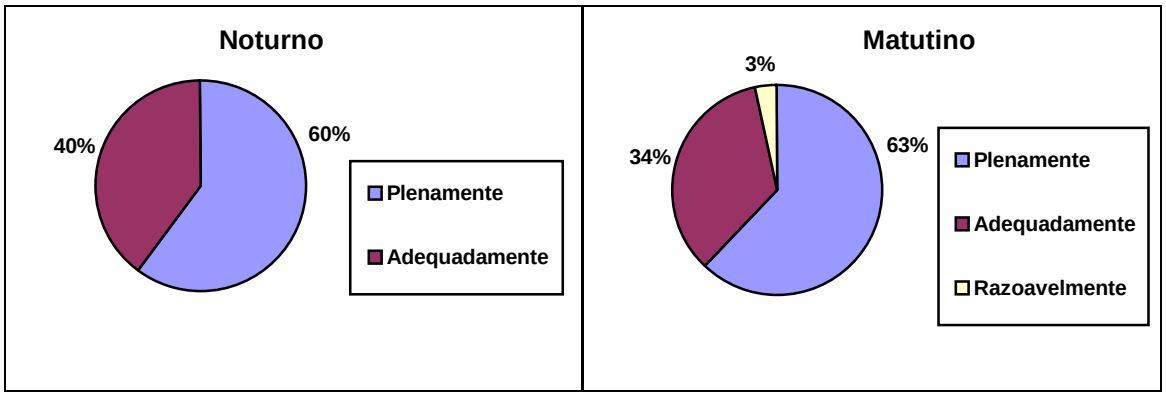
Quadro 56 – Como você classifica a interface da ferramenta?



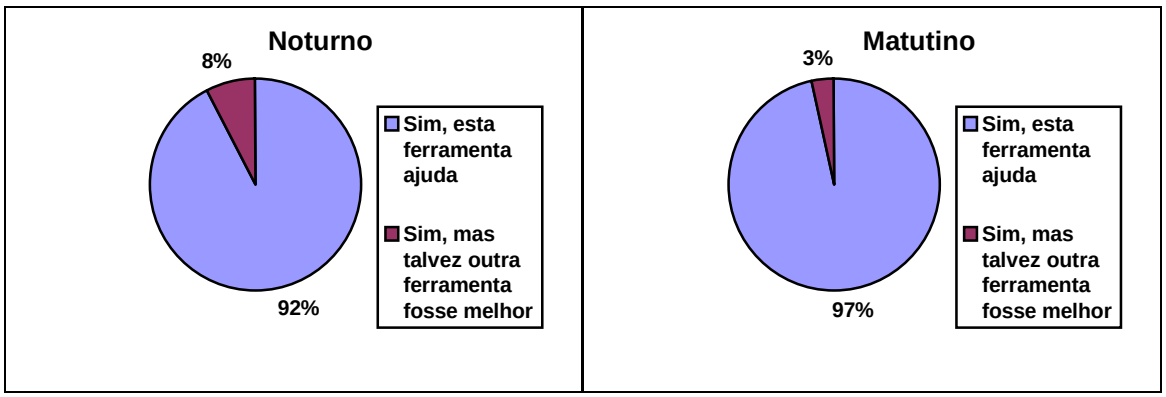
Quadro 57 – As mensagens de erro da ferramenta levaram você a descobrir seu erro?



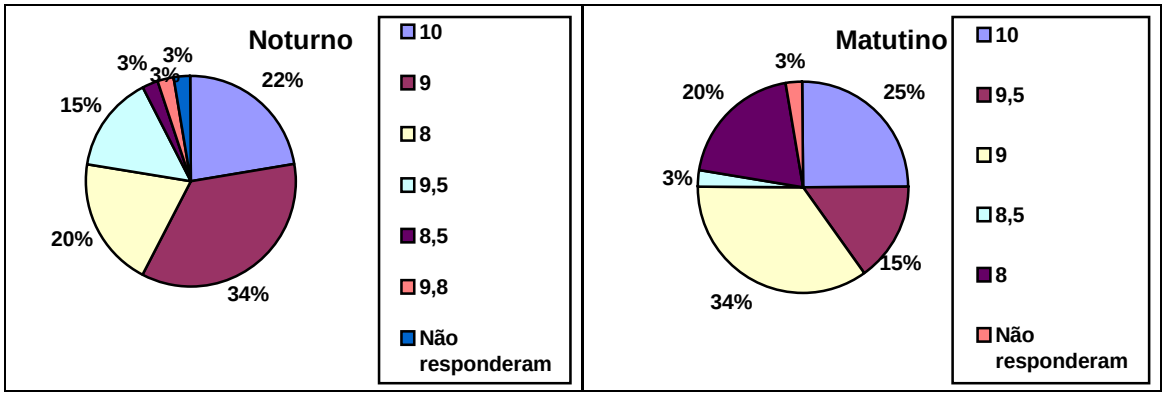
Quadro 58 – Como você classifica a ajuda sobre Portugol da ferramenta?



Quadro 59 – A ferramenta auxiliou no aprendizado da disciplina?

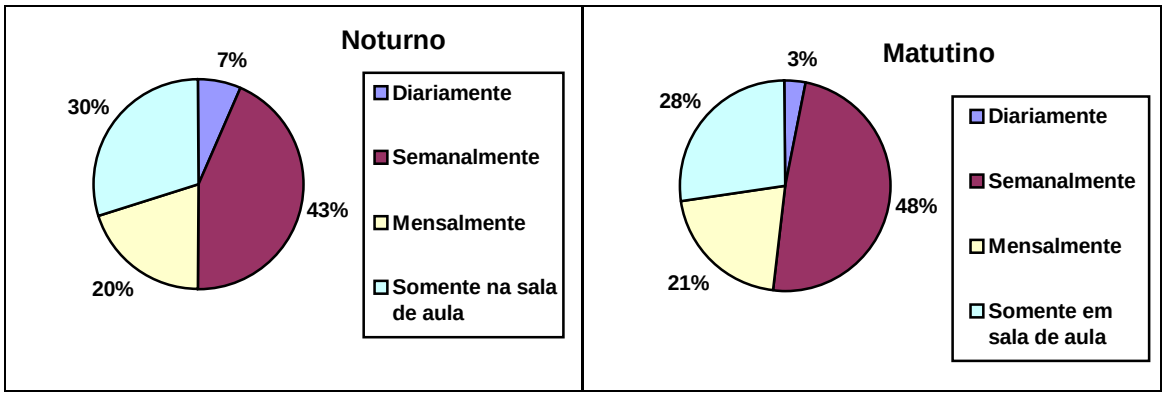


Quadro 60 – Você acha que o uso de uma ferramenta auxilia no aprendizado da disciplina?

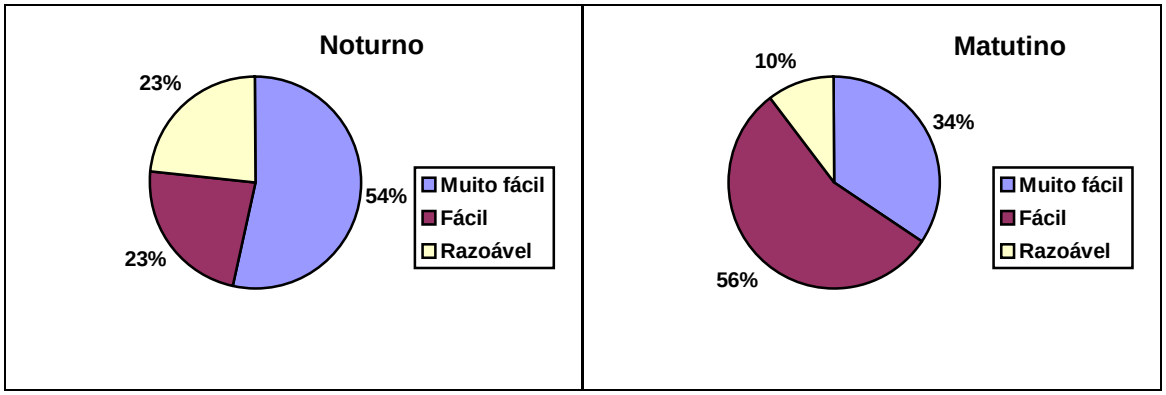


Quadro 61 – Que nota você daria para a ferramenta?

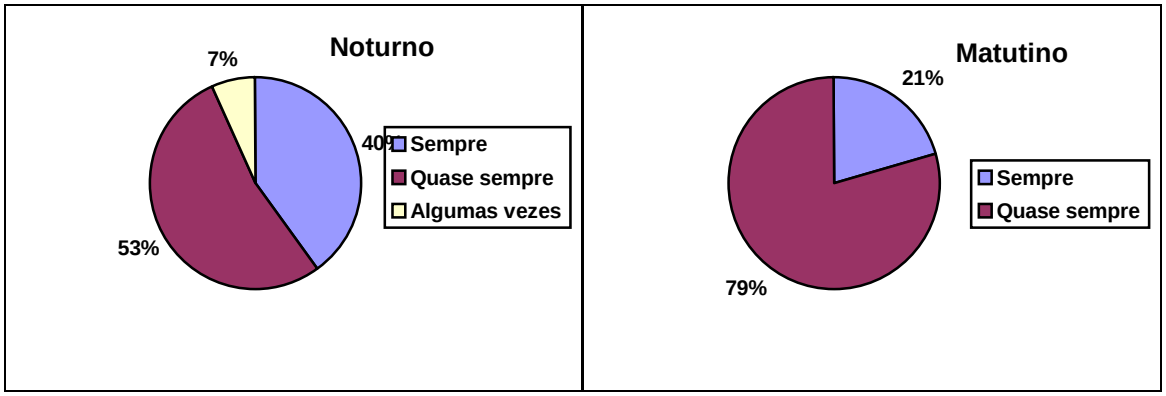
Nos quadros 62 até 71 encontram-se os gráficos com o resultado da avaliação final da ferramenta.



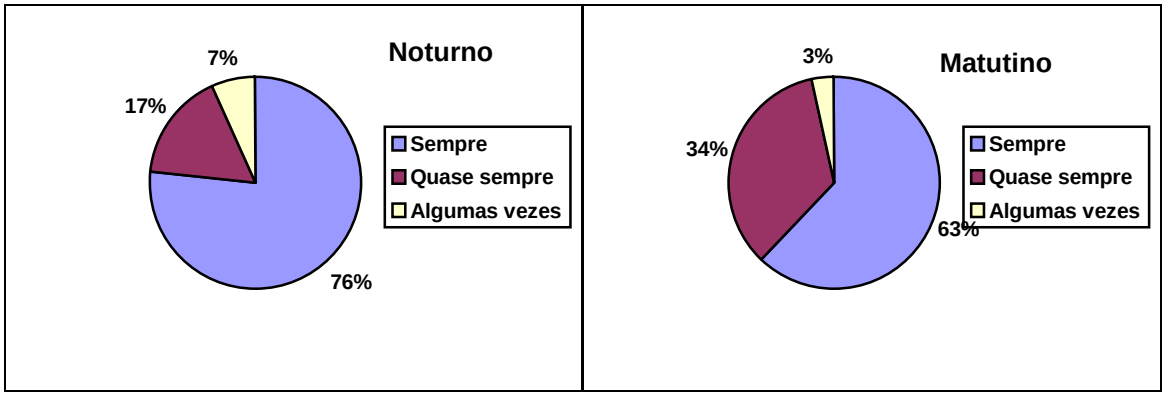
Quadro 62 – Com que freqüência você utilizou a ferramenta?



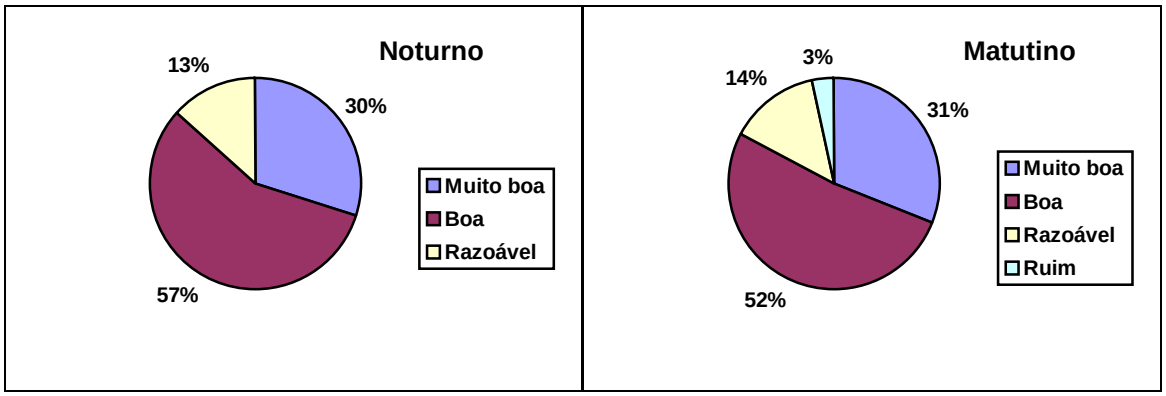
Quadro 63 – Como você classifica a facilidade de uso da ferramenta?



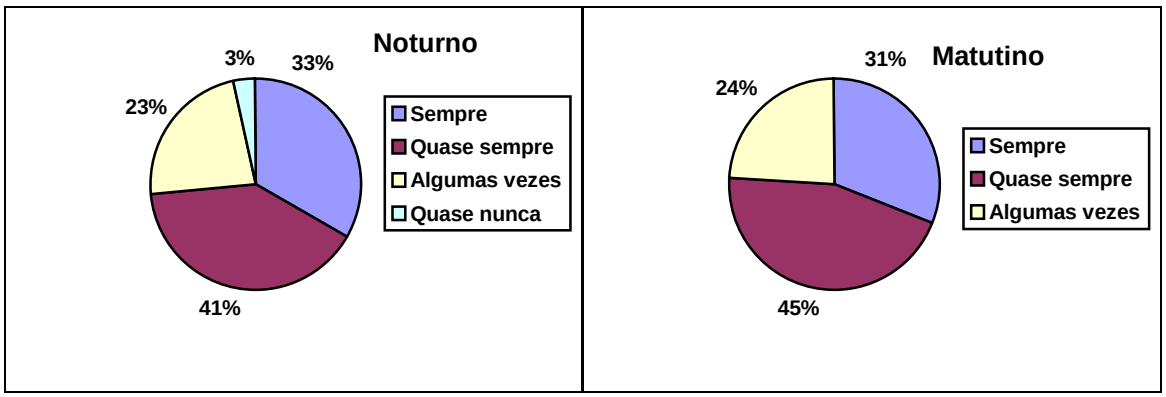
Quadro 64 – A ferramenta responde de acordo com o esperado?



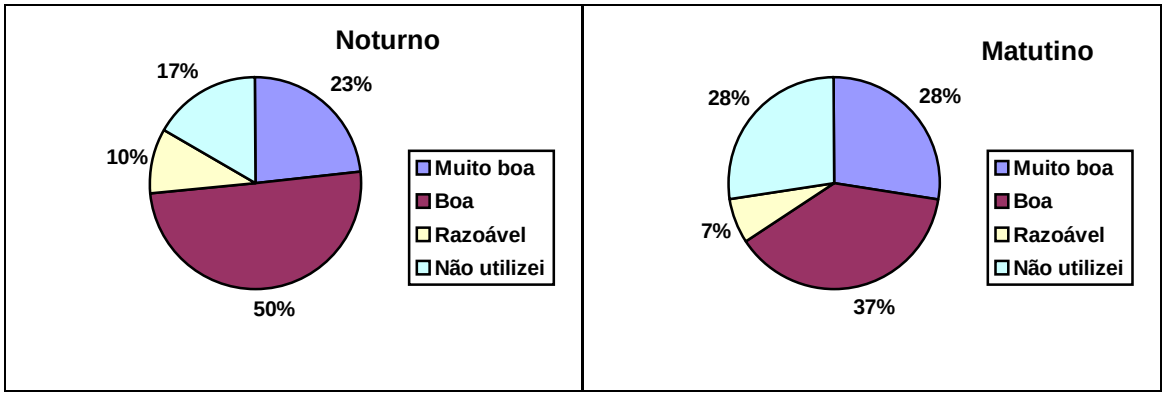
Quadro 65 – Os ícones e atalhos são condizentes com as funções associadas?



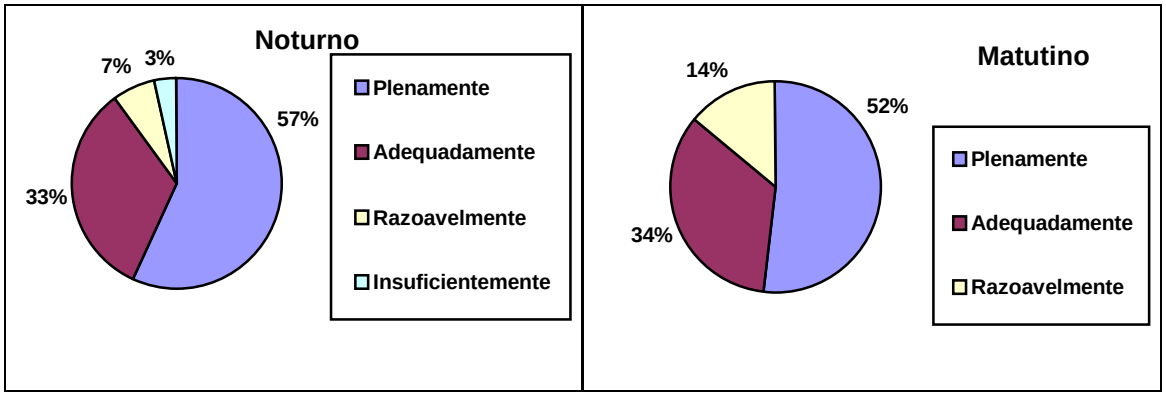
Quadro 66 – Como você classifica a interface da ferramenta?



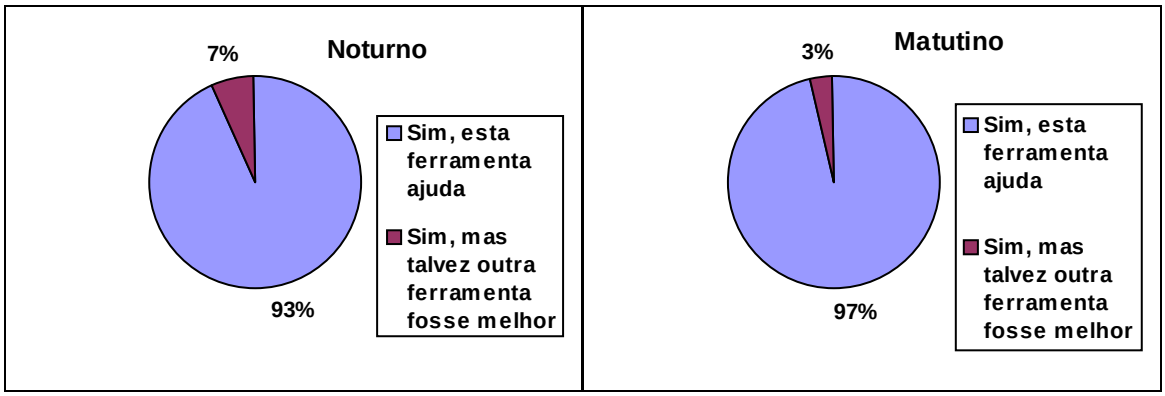
Quadro 67 – As mensagens de erro da ferramenta levaram você a descobrir seu erro?



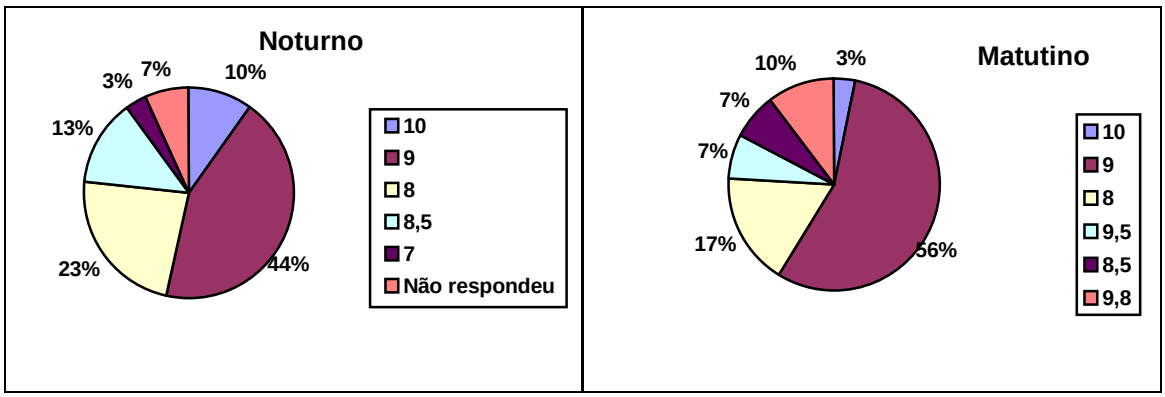
Quadro 68 – Como você classifica a ajuda sobre Portugol da ferramenta?



Quadro 69 – A ferramenta auxiliou no aprendizado da disciplina?



Quadro 70 – Você acha que o uso de uma ferramenta auxilia no aprendizado da disciplina?



Quadro 71 – Que nota você daria para a ferramenta?