UNIVERSIDADE FEDERAL DE ALFENAS
INSTITUTO DE CIÊNCIAS EXATAS
BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

Hiago Borges de Oliveira

UM COMPILADOR, UMA LINGUAGEM DE PROGRAMAÇÃO E UMA MÁQUINA VIRTUAL SIMPLES PARA O ENSINO DE LÓGICA DE PROGRAMAÇÃO, COMPILADORES E ARQUITETURA DE COMPUTADORES

Alfenas, 10 de Fevereiro de 2014.

UNIVERSIDADE FEDERAL DE ALFENAS
INSTITUTO DE CIÊNCIAS EXATAS
BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

UM COMPILADOR, UMA LINGUAGEM DE PROGRAMAÇÃO E UMA MÁQUINA VIRTUAL SIMPLES PARA O ENSINO DE LÓGICA DE PROGRAMAÇÃO, COMPILADORES E ARQUITETURA DE COMPUTADORES

Hiago Borges de Oliveira

Monografia apresentada ao Curso de Bacharelado em Ciência da Computação da Universidade Federal de Alfenas como requisito parcial para obtenção do Título de Bacharel em Ciência da Computação.

Orientador: Professor Luiz Eduardo da Silva

Alfenas, 10 de Fevereiro de 2014.

Hiago Borges de Oliveira

UM COMPILADOR, UMA LINGUAGEM DE PROGRAMAÇÃO E UMA MÁQUINA VIRTUAL SIMPLES PARA O ENSINO DE LÓGICA DE PROGRAMAÇÃO, COMPILADORES E ARQUITETURA DE COMPUTADORES

A Banca examinadora abaixo-assinada aprova a monografia apresentada como parte dos requisitos para obtenção do título de Bacharel em Ciência da Computação pela Universidade Federal de Alfenas.

Professor Gabriel Gerber Hornink

Universidade Federal de Alfenas

Professor Luiz Eduardo da Silva (Orientador)

Universidade Federal de Alfenas

Professor Paulo Alexandre Bressan

Universidade Federal de Alfenas

Alfenas, 10 de Fevereiro de 2014.

"Agradeço todas as dificuldades que enfrentei, se não fosse por elas, eu não teria saído do lugar. As facilidades nos impedem de caminhar. Mesmo as críticas nos auxiliam muito."

Chico Xavier

RESUMO

Nas fases iniciais dos cursos de computação, existe uma dificuldade, por parte dos alunos, para entender a relação entre os programas escritos em uma linguagem de programação e os passos utilizados pelo computador para executar tal programa. Na maioria das vezes, tal dificuldade está associada ao alto nível de abstração utilizado no ensino ou, ainda, à falta de mecanismos para exemplificar tais processos. Já na disciplina de Compiladores, nota-se uma certa dificuldade no aprendizado e falta de motivação do aluno devido à quantidade de técnicas complexas envolvidas na disciplina. Existem ferramentas educacionais cuja proposta é auxiliar no ensino destas disciplinas, porém tais ferramentas limitam-se ao ensino de apenas uma disciplina. Este trabalho apresenta uma ferramenta que oferece um ambiente integrado de desenvolvimento que possibilita ao aluno desenvolver, depurar e executar programas na linguagem Simples. O ambiente inclui um simulador para a Máquina Virtual Simples. O simulador tem o objetivo de exibir de forma gráfica o estado da memória interna e dos registradores da máquina virtual após a execução de cada uma das instruções do código-objeto carregado. Além das funções já citadas, a ferramenta pode, ainda, realizar a execução passo a passo do código Simples mostrando, a cada linha, qual é o código-objeto equivalente.

Palavras-Chave: compiladores, aprendizagem de programação, arquitetura de computadores, simulador didático.

ABSTRACT

In the early stages of computer courses, there is a difficult for the students to understand the relationship between programs written in a programming language and the steps used by the computer to run this program. Most often, this difficulty is associated with a high level of abstraction used in teaching or even the lack of mechanisms to exemplify such processes. In the discipline of compilers, there is a certain difficulty in learning and lack of student motivation due to the amount of complex techniques involved. There are educational tools whose purpose is to assist in the teaching of these subjects, but such tools are limited to teaching only one subject. This paper presents a tool that provides an integrated development environment that enables the student to develop, debug and run programs in Simples language. The environment includes a simulator for the Máquina Virtual Simples. The simulator is intended to graphically display the status of the internal memory and registers the virtual machine after the execution of each instruction object code loaded. Besides the functions already mentioned, the tool can also perform step by step execution of the Simples code showing, every line, which is the object code equivalent.

Keywords: compilers, learning programming, computer architecture, didactic simulator.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIAÇÕES

MVS	Máquina Virtual Simples
Unifal-MG	Universidade Federal de Alfenas

SUMÁRIO

1
Introdução

Neste capítulo são apresentadas a motivação do trabalho e a ideia a ser desenvolvida

Um dos maiores desafios no ensino dos cursos de computação é mostrar como os processos funcionam nas várias camadas do computador. Artigos recentes na área de ensino da computação sugerem que ferramentas educacionais para simulação são fundamentais para o processo de ensino-aprendizagem. Na maioria das vezes, os materiais didáticos utilizam diagramas e *snapshots* para representar um programa em execução. Embora esta abordagem possa ser uma boa forma de exemplificar o conteúdo discutido, os autores consideram que, claramente, esta não é uma maneira eficaz de representar a execução de um programa, uma vez que a execução é um processo dinâmico (ORUÇ, GUNDUZHAN, 2003).

Os programas são descritos por meio de notações, permitindo ao programador definir as tarefas que devem ser realizadas pelo computador sem se preocupar com os detalhes da máquina. Tais notações são chamadas linguagens de programação, entretanto, um programa, primeiro, precisa ser traduzido para um formato que lhe permita ser executado por um computador. Os sistemas de *softwares* que fazem essa tradução são denominados compiladores.

"Colocando de uma forma bem simples, um compilador é um programa que recebe como entrada um programa em linguagem de programação – a linguagem fonte – e o traduz para um programa equivalente em outra linguagem – a linguagem objeto" (AHO, 2008, pag. 1), ou seja, a partir de uma linguagem próxima à linguagem humana (linguagem de alto nível) gera uma linguagem que o computador possa entender (linguagem de baixo nível).

O compilador também deve relatar ao programador a presença de erros léxicos, sintáticos e semânticos no programa-fonte. Existe um grande número de

compiladores, para diversas linguagens, mas as tarefas básicas realizadas por eles são praticamente as mesmas.

Os compiladores foram considerados, por muito tempo, programas difíceis de desenvolver, até que foram descobertas técnicas para o tratamento das mais importantes tarefas que ocorrem durante a compilação. De forma semelhante, foram desenvolvidas ferramentas de *software* e bons ambientes e linguagens de programação. Com estes avanços, um compilador substancial pode ser escrito por alunos em um curso semestral (AHO, 2008).

Para que um aluno de graduação compreenda determinados conceitos na área de compiladores, normalmente o professor da disciplina solicita a elaboração de um compilador simplificado. Porém, devido à alta complexidade dos algoritmos envolvidos, o aprendizado de compiladores sempre representou um grande desafio junto aos alunos.

1.1 Justificativa e Motivação

É evidente nos cursos de computação, a dificuldade de aprendizagem e a falta de motivação do aluno em relação aos fundamentos teóricos das disciplinas de Compiladores, Programação e Arquitetura de Computadores.

Na disciplina de Compiladores, os alunos estudam uma variedade de tópicos sobre a teoria de projeto de um compilador e constroem um compilador para algum subconjunto de uma linguagem, como C ou Java, por exemplo.

Os estudantes de Ciência da Computação acreditam que não aplicarão, no dia a dia de sua vida profissional, os conceitos aprendidos na disciplina. Por esse motivo, consideram o projeto de compiladores irrelevante para sua formação técnica, e assim dedicam menos tempo e esforço ao estudar o projeto de um compilador. Todavia, verifica-se que muitas das técnicas e algoritmos usados por compiladores são amplamente aplicados em outros contextos (HENRY, DEBRAY, 2002).

A construção de compiladores se estende através dos temas de linguagens de programação, arquitetura de computadores, teoria das linguagens, algoritmos, programação e engenharia de software (AHO, 2008), porém os alunos não conseguem perceber com clareza o relacionamento entre elas, talvez, devido à falta de ferramentas ou meios de trabalhar tal relação. Segundo ALMEIDA (2002), a integração dos conceitos dessas disciplinas, com o auxílio de uma ferramenta, facilita o processo de ensino-aprendizagem, culminando na redução de tempo do discente para aprender tal conteúdo.

O presente trabalho tem como motivação a construção de uma ferramenta didática voltada para o ensino de lógica de programação, compiladores e arquitetura de computadores.

1.2 Problematização

Tendo em vista a quantidade de conceitos teóricos e a dificuldade de apresentar claramente a relação direta entre Programação, Arquitetura de Computadores e Compiladores, surge a seguinte questão: é possível construir uma ferramenta que exponha, de forma prática, os conceitos estudados, mostre a relação existente entre tais disciplinas e melhore o processo de aprendizagem por parte dos alunos?

1.3 Objetivos

1.3.1 Gerais

O presente trabalho possui o objetivo geral de criar uma ferramenta didática que possa ser utilizada nas disciplinas de Lógica de Programação, Arquitetura de Computadores e Compiladores.

1.3.2 Específicos

- Construir a ferramenta proposta.

- Utilizar a ferramenta no curso de Ciência da Computação da Unifal-MG.
- Avaliar a aceitação da ferramenta pelos alunos.

1.4 Organização da Monografia

A monografia possui a seguinte organização: no Capítulo 2 é apresentada uma revisão bibliográfica para contextualização dos temas essências ao entendimento do trabalho; no Capítulo 3 é explicado o desenvolvimento da ferramenta; no Capítulo 4 estão os resultados alcançados e no Capítulo 5 é feita a conclusão do trabalho, além de sugestões para trabalhos futuros.

2
Revisão Bibliográfica

Este capítulo apresenta uma revisão bibliográfica sobre o tema abordado na monografia.

A utilização de ferramentas pedagógicas pode ter um grande impacto nas disciplinas de compiladores, arquitetura de computadores e lógica de programação.

Como descrito em VIEIRA (2010) o ensino de conceitos introdutórios de programação costuma apresentar um nível de abstração que prejudica o aprendizado de alunos que apresentam dificuldades em lidar com o raciocínio lógico necessário ao entendimento da lógica de programação.

Pensando nesta dificuldade, os autores propõem uma família de processadores, denominada BIP – Basic Instruction-set Processor, que foi desenvolvida visando proporcionar uma redução da abstração envolvida em conceitos fundamentais da lógica de programação e também apoiar o ensino em disciplinas de fases iniciais do curso de Ciência da Computação. Neste contexto, apresenta-se no trabalho um ambiente de desenvolvimento integrado que possibilita o desenvolvimento, execução e simulação de programas em linguagem Portugol, relacionando-os à arquitetura dos processadores da família BIP.

Ainda segundo VIEIRA (2010), nas fases iniciais de um curso de graduação, os processadores utilizados para o ensino concorrente da lógica de programação e de conceitos de arquitetura de computadores devem facilitar o estabelecimento de relações entre as abstrações lógicas necessárias à programação e sua representação em hardware. Porém, os modelos de processadores tipicamente utilizados são muito abstratos e não permitem estabelecer essas relações.

Uma alternativa seria a utilização de processadores mais detalhados, porém esses processadores são demasiadamente complexos para serem aplicados em disciplinas do primeiro ano, e poucos são os livros texto da área que os descrevem

propiciando uma integração entre a arquitetura do processador e a programação em alto nível.

Ainda no contexto da complexidade de arquiteturas reais, KOWALTOWSKI (1983), explica que traduzir um programa-fonte para uma linguagem de máquina de um computador real não é fácil, porque cada arquitetura de computador apresenta a própria linguagem. O autor definiu uma máquina hipotética a fim de viabilizar a compreensão entre as construções do programa-fonte e sua tradução.

2.1 Linguagens de Programação

Os primeiros computadores tinham os seus programas escritos em uma sequência de zero ou uns, ou seja, em código de máquina. Desenvolver programas desse tipo era um processo cansativo, lento e sujeito a erros (PRICE e TOSCANI, 2001).

Uma linguagem de programação representa a forma de comunicação entre quem programa (o programador) e a máquina (o computador). O desenvolvimento de um programa torna-se mais fácil se a linguagem de programação em uso estiver próxima ao problema a ser resolvido. Esse tipo de linguagem é conhecido como linguagem de alto nível, conforme PRICE e TOSCANI (2001) abordam.

Em 1950, aparecem as linguagens chamadas "simbólicas" ou de "montagem"; cujo intuito era amenizar as intempéries da programação em notação binária. Nas linguagens de montagem as instruções de máquina foram substituídas por mnemônicos. Dessa forma, um programa em linguagem simbólica precisa ser traduzido para a linguagem de máquina antes da execução.

Apenas na década de 60 é que começaram surgir linguagens com notação mais próximas dos programadores (PRICE e TOSCANI, 2001), dentre elas: FORTRAN, PASCAL, ALGOL e COBOL. São classificadas como linguagens procedimentais (um programa especifica uma sequência de passos a serem seguidos para solucionar um problema) e declarativas (divididas em duas classes: funcionais, as quais se baseiam na teoria das funções recursivas; e lógicas, cuja base é a lógica matemática). Nas décadas subsequentes, segundo AHO (2008), as

linguagens de programação aparecem com recursos inovadores no intuito de deixar a programação mais fácil, natural e poderosa.

2.2 Arquitetura de Computadores

Um computador, de forma simplificada, está dividido em software e hardware, que são bastante distantes. Um professor qualificado certamente demonstrará e acentuará a sua ligação, mas nem todos os alunos conseguem reconhecer isso. Uma das possíveis formas de associar hardware ao software é demonstrar visualmente o que acontece "dentro" do computador (EREMIN, 2005).

Sabe-se que em "Arquitetura e Organização de Computadores" são ministrados conteúdos de sistemas computacionais, dentro dos quais há a separação entre arquitetura e organização do computador.

O termo "arquitetura de computadores", segundo STALLINGS (2003), aplica-se aos atributos de um sistema que exerce grande influência sobre a execução lógica de um programa. O termo "organização de um computador" refere-se aos componentes operacionais que descrevem as particularidades da sua arquitetura. Exemplos de atributos de arquitetura incluem o conjunto de instruções, o número de bits usados para representar tipos de dados, os mecanismos de Entrada e Saída e as técnicas de endereçamento à memória. Atributos de organização envolvem detalhes de hardware visíveis ao programador, como sinais de controle, interfaces entre o computador e periféricos e tecnologia de memória utilizada (STALLINGS, 2003). O conjunto de instruções é uma coleção de diferentes informações que a Unidade Central de Processamento é capaz de executar.

Vários aspectos na definição e implementação da arquitetura de um computador são influenciados pelas características do conjunto de instruções, como por exemplo, a estrutura e a complexidade (AHO, 2008).

Para que se tornem operacionais, os programas escritos em linguagens de alto nível devem ser traduzidos para linguagem de máquina. Essa conversão é realizada por meio de sistemas especializados, os compiladores. Desde a década de

40, as arquiteturas de computadores evoluíram. Assim, os projetistas de compiladores tiveram não apenas de acompanhar novos recursos das linguagens, mas também desenvolver algoritmos de tradução que tirassem o máximo de proveito das capacidades de hardware (AHO, 2008).

Os compiladores podem propagar o uso de linguagens de alto nível, diminuindo o custo adicional da execução dos programas escritos nessas linguagens. Os compiladores também são responsáveis por efetivar o uso das arquiteturas de computador de alto desempenho nas aplicações dos usuários. Na prática, o desempenho de um sistema de computação é muito dependente da tecnologia de compilação (AHO, 2008).

Como foi dito, o compilador traduz uma linguagem-fonte descrita em uma linguagem de alto nível para uma linguagem-objeto equivalente em código de máquina para um processador. Geralmente, um compilador não produz diretamente o código de máquina que são formados por uma sequência de bits, mas sim um programa em linguagem simbólica, o Assembly, semanticamente equivalente ao programa em linguagem de alto nível, entretanto, traduzir para a linguagem de máquina de um computador real é uma tarefa trabalhosa. Ao invés de traduzir, a tendência é a utilização de uma máquina hipotética no desenvolvimento de um compilador didático.

Diversos compiladores utilizam de máquinas hipotéticas que geram um código intermediário entre a linguagem de máquina e a linguagem alto nível. Este código posteriormente será traduzido em linguagem de máquina por meio do montador (assembler). Este formato deixa claro e facilita na compreensão das estruturas compiladas, pois dispõe de instruções com formas mais simplificadas. Sendo assim, o aluno irá adquirir maior conhecimento de como as estruturas da linguagem são transformadas para instruções próximas à da linguagem de máquina.

2.3 Compiladores

No meio do século passado, foram implementados os primeiros compiladores para a linguagem FORTRAN. À época, as técnicas de projeto e implementação de

linguagens ainda não estavam muito desenvolvidos. No decorrer dos anos 50, os compiladores foram considerados programas difíceis de implementar. A partir daí foram descobertas técnicas sistemáticas para o tratamento das mais importantes tarefas que ocorrem durante a compilação. Da mesma forma, foram desenvolvidas ferramentas de software, boas linguagens de implementação e ambientes de programação (AHO, 2008).

Existem basicamente duas partes no processo de compilação, a análise e a síntese. A análise divide o programa fonte em partes constituintes e cria uma representação intermediária do programa fonte. Tem como principal objetivo detectar se o programa fonte está sintaticamente ou semanticamente incorreto e então gerar mensagens que direcionam o usuário a tomar as devidas correções. A parte de análise também coleta informações sobre o programa fonte e as armazena numa estrutura de dados chamada "tabela de símbolos", que é passada adiante junto com a representação intermediária para a parte de síntese (AHO, 2008).

A parte de síntese é responsável pela construção do programa objeto, tendo como base as informações armazenadas na tabela de símbolos que foi gerada na parte de análise.

Um compilador, portanto, é composto de várias fases, cada qual, com sua função específica. A interação destas fases tem como objetivo transformar o programa fonte em uma representação para outra denominada de código objeto (AHO, 2008). Para AHO (2008), o processo de compilação é desenvolvido como uma sequência de seis fases, representadas na figura a seguir.

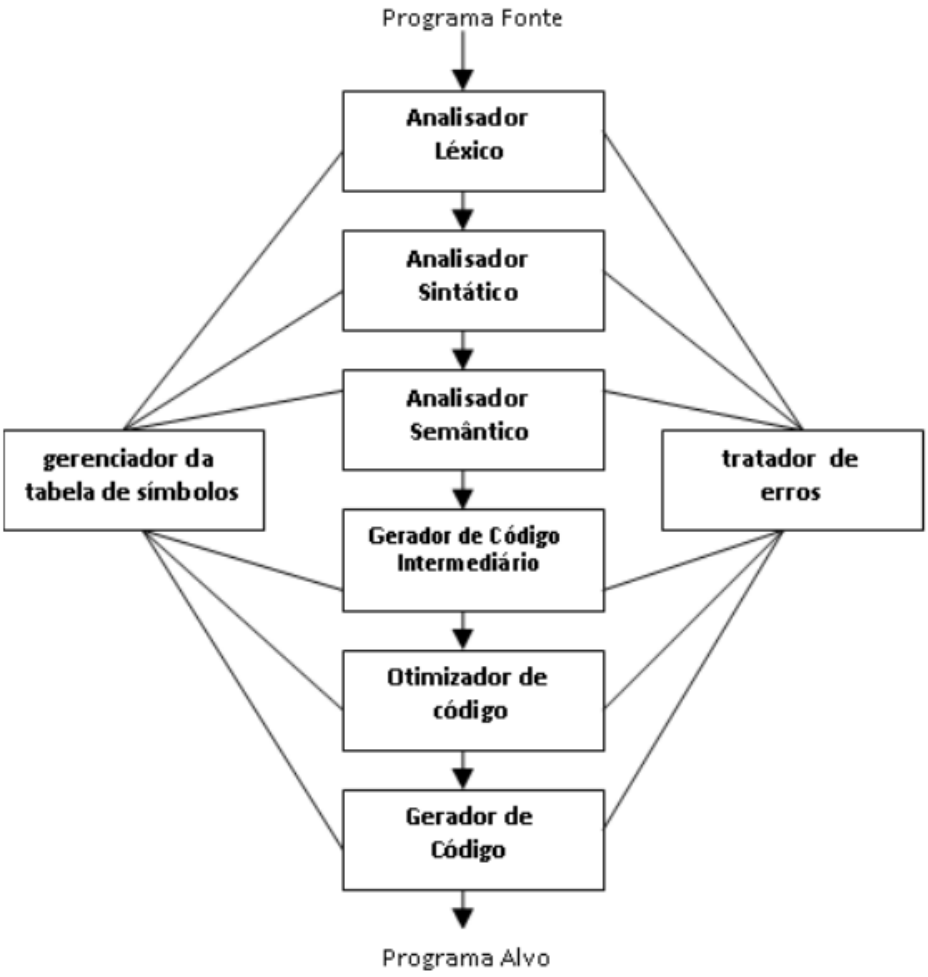


Figura 1 - Fases de um compilador (AHO, 2008)

2.3.1 Analisador Léxico

O analisador léxico tem a função de ler o programa produzindo como resultado uma sequência de *tokens*. Um *token* é composto por duas partes: nome e valor. O nome é um símbolo utilizado na análise sintática e o valor é um apontador para a tabela de símbolos, fazendo referência ao *token* (AHO, 2008). Os *tokens* são definidos por meio de expressões regulares, logo o analisador léxico pode ser visto como um autômato finito determinístico.

Além de converter programas em sequências de *tokens*, os analisadores léxicos também podem eliminar espaços em branco e comentários.

2.3.2 Analisador Sintático

Após o programa ser convertido para uma sequência de *tokens*, o próximo passo é verificar se a sequência é válida, construindo, assim, uma árvore sintática.

O analisador sintático verifica se a sequência de *tokens* é uma sentença válida da linguagem-fonte, caso não seja, um erro sintático é gerado (AHO, 2008). A análise sintática utiliza gramáticas livres do contexto.

2.3.3 Analisador Semântico

Feita a análise sintática do programa, parte-se para a análise semântica. Um programa pode estar sintaticamente correto, porém pode não fazer sentido.

Os aspectos sensíveis ao contexto são tratados pelo analisador semântico. Um exemplo de regra semântica seria: "apenas variáveis do tipo inteiro podem ser utilizadas nas operações de soma".

A análise semântica é feita por meio da manutenção da tabela de símbolos (CRESPO, 1998 apud SCHNEIDER, PASSERINO, OLIVEIRA, 2005) e da pilha semântica (APPEL, 1998 apud SCHNEIDER, PASSERINO, OLIVEIRA, 2005).

3
Desenvolvimento da Ferramenta

Este capítulo detalha o desenvolvimento da ferramenta proposta, explicando cada uma das partes que compõem o projeto.

O objetivo deste trabalho é propor e construir uma ferramenta para o ensino das disciplinas de compiladores, lógica de programação e arquitetura de computadores. Apesar de o produto proposto pelo projeto ser chamado sempre de "ferramenta", é necessário ressaltar que o resultado almejado não é apenas um *software,* e sim um conjunto de *softwares*, onde alguns são totalmente desacoplados dos demais, porém, trabalham em conjunto para atender os objetivos propostos.

O projeto propõe o desenvolvimento de um compilador, de uma linguagem de programação e de uma máquina virtual. Primeiramente, foi proposta a linguagem de programação nomeada de Simples. Como já dito em seu nome, a linguagem Simples surgiu para ser uma linguagem simples, de fácil aprendizado, com os comandos da linguagem assemelhando-se às instruções, escritas no nosso idioma nativo, o português, que seriam utilizadas no dia a dia para descrever um conjunto de passos qualquer, uma receita de bolo, por exemplo.

Na definição da linguagem, a simplicidade e a facilidade de aprendizado foram o foco, porém o projeto não deixou de fora os comandos necessários para desenvolver programas complexos, sendo eles comandos de repetição, desvio condicional, atribuição, avaliação de expressões lógicas e aritméticas, leitura e escrita.

Proposta a linguagem Simples, fez-se necessária à idealização de um *software*, capaz de converter os programas escritos em linguagem de programação Simples em programas escritos em algum tipo de linguagem que o computador

pudesse entender e, consequentemente, executar o programa. Para este fim é utilizado um compilador.

O compilador, que transforma a linguagem de programação Simples em um código que pode ser entendido por uma máquina, foi nomeado de Compilador Simples. O Compilador Simples foi idealizado para, além de possibilitar a execução de programas escritos em linguagem Simples, ser um compilador educacional, uma ferramenta didática, alvo de estudos por partes dos alunos da disciplina de compiladores.

Preencher o abismo que separa uma linguagem de programação de alto nível, que se assemelha a nossa linguagem natural, e uma linguagem de máquina é tarefa do compilador. Para mensurar a distância entre as duas pontas, seria necessário, primeiro, conhecer, a fundo, os dois a serem ligados. Em um dos lados, a linguagem Simples, proposta no escopo deste trabalho, de outro uma linguagem de máquina. Escolhendo uma linguagem de máquina real, já existente, seria necessário estudá-la a fundo, antes de construir um compilador para tal linguagem. Tendo em vista que as arquiteturas dos *hardwares* atuais são fruto de anos de pesquisa, almejando um desempenho ótimo, supõe-se que uma linguagem de máquina de um computador real seja muito completa. Não sendo viável a criação de um compilador que converta a linguagem de programação Simples em uma linguagem de máquina real, torna-se evidente a necessidade da criação de uma máquina hipotética, nomeada Máquina Virtual Simples.

A Máquina Virtual Simples (MVS) é um *software* que simula o funcionamento de uma máquina capaz de executar um determinado conjunto de instruções, propostas no projeto da MVS. A execução de uma instrução pode ser entendida como um comando que causa uma alteração no estado da máquina, podendo esta ser uma alteração dos contadores ou de uma posição de memória. Posteriormente, o funcionamento da MVS será detalhado.

Sendo a máquina para a qual será gerado código, uma máquina hipotética, nada mais do que um *software* com a função de simulador, surgem algumas possibilidades bastante interessantes do ponto de vista didático: além de simplesmente executar uma sequência de comandos e, ao final do programa, apresentar um resultado, é possível exibir, de maneira gráfica, o estado interno na máquina após a execução de cada instrução, permitindo aos utilizadores da

ferramenta observar o impacto de cada instrução executada sobre o estado da máquina. É esperado que, desta forma, a ferramenta possa facilitar o aprendizado dos alunos na disciplina de arquiteturas de computadores além de auxiliá-los no entendimento da relação entre os algoritmos escritos, a linguagem de máquina gerada e os recursos que a máquina oferece. Descritos os componentes que compõem a ferramenta, o porquê foram propostos e seus respectivos objetivos, nas seções seguintes será detalhado cada um deles.

3.1 A Linguagem de Programação Simples

Para os alunos, nos períodos iniciais, de um curso de ciência da computação e afins, espera-se que uma das primeiras habilidades a serem trabalhadas é a habilidade da lógica de programação, exercitada por meio do desenvolvimento de algoritmos, que podem ser vistos como um conjunto de passos para cumprir um objetivo.

Caso o professor de uma disciplina de algoritmos não possuísse nenhuma ferramenta que considerasse ideal para iniciantes e julgasse que ensinar uma linguagem de programação (dentre as mais utilizadas) agregaria uma dificuldade a mais para os graduandos, pois além de aprender e exercitar a lógica de programação teriam de aprender as características da linguagem em questão, o mais provável seria que as atividades do curso fossem feitas, a princípio, de forma textual.

Fazer algoritmos, utilizando uma linguagem próxima ao nosso idioma, a língua portuguesa, em um papel talvez seja uma boa maneira de exercitar a criação de algoritmos que resolvem problemas específicos, entretanto, é esperado que, caso os alunos pudessem desenvolver tais algoritmos, de forma semelhante à descrita anteriormente, porém, utilizando um ambiente computacional, seria mais interessante, além de mais motivador.

Com o ambiente computacional, os alunos teriam a possibilidade de executar tais algoritmos e verificar seus resultados de forma interativa. O ambiente computacional possibilitaria, também, aos alunos aprender com seus próprios erros. Em uma folha de papel, um pequeno erro pode não ser percebido, já em um ambiente computacional, um pequeno equívoco pode ter um grande impacto. Caso

haja um erro na forma de descrever um comando, o computador interpretaria como uma construção inválida e informaria ao usuário que não pôde interpretar. Já no caso de um erro de lógica, ou seja, o conjunto de passos descrito não realizará exatamente o que o aluno espera e será fácil observar que o resultado da execução não é o almejado.

3.1.1 Definição da Linguagem Simples

A linguagem Simples possui os seguintes tipos de comandos: desvio condicional, laço de repetição, comandos de leitura e escrita, comandos de atribuição; e os seguintes tipos de expressões: lógicas, aritméticas e relacionais. O projeto inclui também variáveis do tipo inteiro e do tipo lógico.

No momento da definição da linguagem, as características de um programa Simples foram influenciadas pela linguagem de programação C. Os tipos de comentários presentes da linguagem (vistos mais a diante), por exemplo, são exatamente os mesmos presentes na linguagem C.

3.1.2 Definição de um programa

Os programas descritos na linguagem de programação Simples devem possuir, no mínimo, uma característica básica, sendo ela a definição do corpo do programa. O corpo do programa é constituído por algumas palavras que definem as seções do programa. Na figura a seguir temos a definição do menor programa válido escrito em linguagem simples.

```
1   /*
2    *  autor: Nome_do_Autor
3    */
4   programa nome_programa
5
6       //declaração de sub programas (ou não)
7
8       //declaração de variáveis (ou não)
9   inicio
10      //comandos (ou não)
11  fimprograma
```

Figura 2 - Menor programa válido da linguagem Simples

Observa-se na Figura 2, um programa escrito na linguagem de programação Simples é composto por duas seções. A primeira seção é definida pelas palavras "programa", seguida por um identificador que representa o nome do programa, e "inicio". Nesta seção é possível declarar subprogramas e variáveis, ambos serão explicados posteriormente. A segunda seção é definida pelas palavras "inicio" e "fimprograma". A segunda seção é utilizada para definir os comandos executados pelo programa. Os comandos serão explicados posteriormente.

Ao observar a Figura 2 que contém o menor programa possível, observam-se algumas palavras escritas em verde que são ignoradas ao executar o programa porque são comentários. Em uma linguagem de programação, os comentários são trechos nos quais o programador descreve algo ou faz alguma observação que julgue relevante para o entendimento do programa.

Na linguagem Simples existem dois tipos de comentário: comentário de linha e comentário de bloco. Para fazer comentários é necessário utilizar símbolos especiais que definem seu início e o seu fim. No caso de um comentário de linha, é necessário utilizar apenas um símbolo para denotar apenas o início do comentário, o término do comentário será o final da linha. O símbolo utilizado pelo o comentário de linha é formado por duas barras ("//"). Já o comentário de bloco, permite que o desenvolvedor utilize múltiplas linhas para descrever o que julgar necessário. Para marcar o início e o fim de um comentário de bloco são utilizados dois símbolos distintos, sendo eles, respectivamente, uma barra e um asterisco ("/*") e um asterisco e uma barra ("*/"). Abaixo temos alguns exemplos de comentários válidos.

```
1   /* Início de um comentário de bloco
2
3    Parte ignorada pelo programa
4
5   fim de um comentário de bloco   */
6
7   programa nome_programa
8
9       // Comentário de linha: a próxima linha faz parte do programa!
10  inicio
11
12  fimprograma //A parte que antecede o início de um comentário faz parte do programa!
```

Figura 3 - Exemplos de comentários presentes na linguagem Simples

3.1.3 Variáveis

A linguagem Simples fornece a utilização de variáveis do tipo lógico e do tipo inteiro. De uma forma bastante simplificada, pode-se definir uma variável como uma posição de memória utilizada para armazenar um valor que pode ser alterado ou consultado ao longo da execução do programa, sendo a existência da memória uma questão transparente para o programador. Uma variável apresenta duas características fundamentais: identificador e tipo.

Um identificar deve ser único e é utilizado para fazer referência a variável. Na linguagem Simples, foi definido que os nomes utilizados como identificadores devem seguir a seguinte norma: deve começar por uma letra e seguida, ou não, por uma combinação qualquer de letras, números e travessão (*underline*). São exemplos de identificadores válidos: nome, idade, soma_idades, telefone_1. A linguagem Simples, não é *case-sensitive*, o que quer dizer que não há diferenciação entre letras maiúsculas e minúsculas, ou seja, em um programa, os identificadores ab, AB, Ab e aB serão considerados iguais.

O tipo de uma variável está associado aos possíveis valores que podem ser armazenados. No caso da linguagem Simples, existem os tipos inteiro e lógico. Uma variável ser do tipo inteiro significa que apenas valores pertencentes ao conjunto dos números inteiros podem ser armazenados.

Seguindo o mesmo raciocínio, uma variável do tipo lógico pode armazenar apenas valores do tipo lógico. O conjunto dos valores lógicos é composto por apenas dois valores: verdadeiro e falso. Na linguagem simples, foi definido que os valores verdadeiro e falso são representados pelas letras maiúsculas V e F, respectivamente.

Para utilizar uma ou mais variáveis no corpo do programa é necessário especificar o tipo e os identificadores das variáveis em uma seção específica. Tal processo é chamado de declaração das variáveis. Abaixo segue o exemplo de um código válido que declara variáveis de diferentes formas:

```
1   programa nome_programa
2
3       //Declaração de Variáveis
4       inteiro idade       //variavel do tipo inteiro
5       logico eh_maior_18 //variavel do tipo logico
6
7       //declaração de multiplas variáveis do tipo inteiro
8       inteiro soma valor numero
9
10      //declaração de múltiplas variáveis do tipo lógico
11      logico esta_chovendo eh_um_aluno
12
13  inicio
14      //seção de comandos que
15      //podem utilizar as variáveis
16  fimprograma
```

Figura 4 - Exemplo de declaração de variáveis

3.1.4 Comandos

Após definir a estrutura mínima do programa e declarar as variáveis, se necessário, é hora de definir quais as ações a serem executadas. O comportamento ou o funcionamento do programa é descrito através de uma sequência de comandos que devem estar depois da palavra "inicio" e antes da palavra "fimprograma". Existem diferentes tipos de comandos.

3.1.4.1 Comandos de entrada e saída

Os comandos de entrada e saída são responsáveis pela interação do programa com o usuário. Sem estes, não seria possível definir quais são os valores de entrada para o programa e nem seria possível visualizar qual foi o resultado obtido.

O comando de entrada permite ao usuário definir qual será o valor de uma variável, enquanto o comando de saída é utilizado para exibir na tela qual é o valor armazenado em uma variável em um dado momento. Para representar os comandos, foram escolhidas as palavras leia e escreva.

```
1
2   programa nome_programa
3
4       inteiro idade
5
6   inicio
7
8       leia idade
9       escreva idade
10
11  fimprograma
```

Figura 5 - Utilização dos comandos de entrada e saída.

No programa presente na Figura 5, há um comando de leitura na linha oito e um comando de escrita na linha nove. Para ser utilizado, o comando de leitura deve ser seguido por uma variável. Ao ser executado, o comando, através de uma interface com o usuário, deve solicitar que um valor seja digitado. O valor informado será armazenado na variável "idade", neste caso. Já o comando de escrita, presente na linha nove, para ser utilizado, deve ser seguido por uma variável ou por uma expressão (explicada mais adiante). Sua função é exibir na tela o valor atual de um elemento, neste caso a variável idade. Neste exemplo o programa solicita a leitura de um valor, armazena o valor na variável idade e, em seguida, exibe na tela o mesmo valor informado, afinal, nenhuma operação foi realizada sobre a variável idade.

3.1.4.2 Comando de atribuição

Além do comando de leitura, que permite ao usuário informar o valor que a variável deve assumir, existe uma outra forma de alterar o conteúdo de uma variável: através do comando de atribuição. Enquanto o comando de leitura permite ao usuário definir os valores às variáveis, normalmente no início do programa, o comando de atribuição permite que as variáveis assumam os valores dos cálculos, ou operações, realizados ao longo do programa.

A linguagem Simples possui apenas um tipo de comando de atribuição. Um comando de atribuição é composto por uma variável do lado esquerdo e por um valor, expressão ou outra variável do lado direito, sendo a definição de direita e esquerda relativas ao operador de atribuição. O operador de atribuição escolhido é

uma seta apontada para esquerda, representando que o valor do lado direito será armazenado na variável do lado esquerdo. Para "escrever" a seta de forma textual formar utilizados os símbolos menor (<) e subtração (-). Apresenta-se na Figura 6 algumas possíveis atribuições.

```
1   programa atribuicoes
2
3       inteiro a b c
4       logico l1 l2 l3
5   inicio
6       a <- 1
7       b <- (1+1) * 2
8       c <- a -b +10
9
10      l1 <- V
11      l2 <- F
12      l3 <- l1 ou l2
13
14
15  fimprograma
```

Figura 6 - Utilização do comando de atribuição

3.1.4.3 Comando de desvio condicional

Os desvios condicionais são utilizados para alterar o fluxo de execução de um programa, dando ao programador a possibilidade de tratar cenários distintos de diferentes formas.

A execução de um programa consiste na execução sequencial de todos os comandos que compõem o programa, porém existem alguns tipos de comandos que podem alterar a ordem de execução. Na linguagem Simples, foi implementado apenas um tipo de desvio condicional, composto por um teste lógico e dois blocos de comandos. Caso o teste lógico tenha um resultado "verdadeiro" o primeiro bloco de código será executado, caso seja falso, apenas o segundo bloco será executado. É importante observar que, sem exceção, apenas um dos dois blocos será executado. Na Figura 7, tem-se a estrutura de um desvio condicional:

```
10      se valor_verdadeiro entao
11          /*
12           *  Bloco 1
13           */
14      senao
15          /*
16           * Bloco 2
17           */
18      fimse
```

Figura 7 - Estrutura de um comando de desvio condicional

Como se pode observar na Figura 7, a estrutura do comando de desvio condicional utiliza a palavra "se", seguida de uma variável, ou expressão. Esta parte representa o teste a ser realizado. O teste resultará em um valor verdadeiro ou em um valor falso. Caso o resultado seja um valor verdadeiro, o bloco 1, compreendido entre as palavras "entao" e "senao" será executado. Caso contrário, o bloco 2, delimitado pelas palavras "senao" e "fimse" será executado.

Um bloco de comandos pode ser composto por vários comandos, inclusive outros comandos de desvio condicional, por apenas um comando ou, ainda, pode ser um bloco vazio, sem nenhum comando.

Como exemplo da utilização do comando de desvio condicional, na Figura 8 é exemplificado um programa que imprime o maior número digitado pelo usuário.

```
1   programa testa_maior
2
3       inteiro a b
4   inicio
5       leia a
6       leia b
7
8       se a > b entao
9           escreva a  // a é maior que b
10      senao
11          escreva b // a não é maior que b
12      fimse
13
14  fimprograma
```

Figura 8 - Programa utilizado para dizer qual é o maior número

3.1.4.4 Comando de repetição

O comando de repetição, também conhecido como laço de repetição, tem o objetivo de fazer com que um bloco de comandos seja executado por um número determinado de vezes.

A linguagem Simples disponibiliza apenas um comando de repetição, que possui uma condição de permanência e um bloco de comandos a ser executado enquanto a condição de permanência for válida. De maneira semelhante ao comando de desvio condicional, o bloco de comandos do comando de repetição pode ter vários, um ou nenhum comando, porém, mais à frente, será mostrado que a utilização de um bloco de comandos vazio traz um problema. Na Figura 9, pode-se observar um exemplo de comando de repetição.

```
1   programa repeticao
2
3       inteiro i
4   inicio
5
6       i <- 1
7
8       enquanto i < 11 faca
9
10          escreva i
11          i <- i + 1
12
13      fimenquanto
14
15
16  fimprograma
```

Figura 9 - Exemplo de comando de repetição

No programa apresentado na Figura 9, há um comando de repetição na linha oito. O comando de repetição é definido através da palavra “enquanto”, seguido da condição de permanência do laço que pode ser uma variável lógica ou uma expressão. Em seguida, há o bloco de comandos a ser executado. O bloco é delimitado pelas palavras “faca” e “fimenquanto”.

Na figura em questão, aparece um programa que escreve na tela todos os números de um a dez. A condição de permanência do comando de repetição é que a variável i seja menor do que o número onze. Dentro do bloco de comandos temos

uma linha onde, através de um comando de atribuição, a variável i é incrementada em uma unidade a cada iteração, até que a variável apresente um valor maior do que onze, não satisfazendo mais a condição de permanência do comando. Caso, o comando de atribuição que alterara o valor da variável i não existisse, o comando de repetição seria executado infinitamente, pois a condição de permanência seria sempre válida. Por este motivo, ao criar comandos de repetição com um bloco de comandos vazio, resulta em um problema.

3.1.5 Expressões

Nos comandos apresentados anteriormente, em algumas situações, tanto variáveis quanto expressões eram aceitas, como por exemplo, no teste condicional ou na condição de permanência do laço de repetição. Tal fato ocorre porque, ao encontrar uma expressão, o programa a resolve para depois prosseguir com a avaliação do comando, ou seja, durante a execução do programa as expressões são reduzidas a apenas um valor para que então este seja utilizado do comando.

```
1   programa expressoes
2
3       inteiro i
4   inicio
5
6       i <- 1+1
7       escreva i
8
9
10  fimprograma
11
```

Figura 10 - Exemplo da utilização de uma expressão

No programa presente na Figura 10, mais precisamente na linha seis, pode-se obervar a utilização de uma expressão em um comando de atribuição. Neste caso, ao executar o comando de atribuição, o programa resolverá a expressão aritmética de soma para que seu resultado possa ser utilizado. Já na figura a seguir, é possível ver um exemplo em que o resultado de uma expressão é utilizado para definir qual ação o comando de desvio condicional deve tomar.

```
1   programa expressoes
2
3       inteiro i
4   inicio
5
6       leia i
7
8       se i > 0 entao
9           escreva i
10      senao
11          escreva 0
12      fimse
13
14
15  fimprograma
```

Figura 11 - Utilização de expressão no comando de desvio condicional

Na linguagem de programação Simples existem os seguintes tipos de expressões: lógicas, aritméticas e relacionais.

3.1.5.1 Expressões aritméticas

Uma expressão aritmética é uma sentença, onde constantes, neste caso os números inteiros, e variáveis, através dos operadores aritméticos, produzem como resultado um valor inteiro. A linguagem Simples disponibiliza os operadores de soma, subtração, multiplicação e divisão, sendo a precedência das operações a mesma adotada na matemática.

Para alterar a precedência das operações, pode-se utilizar parênteses. A seguir estão representadas algumas expressões aritméticas e os respectivos símbolos utilizados para representar cada operador.

```
1   programa expressoes_aritmeticas
2
3       inteiro a b c
4   inicio
5
6       a <- 1
7       b <- 2
8       c <- 3
9
10      escreva a + b // adição
11      escreva a - b // subtração
12      escreva a * b // multiplicação
13      escreva b div a //divisão
14
15      escreva c * 10 // multiplicação de uma variável por uma constante
16
17      escreva (a + b) div c // adição primeiro e depois divisão
18
19  fimprograma
```

Figura 12 - Expressões Aritméticas

Após observar as expressões aritméticas apresentadas, observa-se que o operador de divisão adotado no projeto não é um símbolo, como os demais. Um operador diferenciado foi escolhido como uma forma de dizer ao programador que a operação de divisão disponibilizada traz um diferencial em relação à operação de divisão normal. Como resultado de uma divisão em um programa escrito na linguagem Simples, são produzidos apenas números inteiros, no caso de divisões que possuam uma parte fracionária, apenas a parte inteira do resultado será considerada.

3.1.5.2 Expressões relacionais

As expressões relacionais avaliam se a relação, definida por um operador é verdadeira ou falsa. Talvez, a partir do uso dos operadores relacionais, a aplicação do uso dos valores lógicos se torne mais evidente.

No projeto da linguagem Simples foram adotados os operadores relacionais: maior, menor e igual, representados pelos símbolos “>”, “<” e “=”. Mais operadores relacionais poderiam ter sido implementados, porém a complexidade adicional dos novos operadores não traria grandes benefícios a linguagem, uma

vez que, quando combinados aos operadores lógicos, é possível simular diversos tipos de relações entre constantes e variáveis.

As relações de maior e menor são utilizadas entre variáveis e constantes inteiras, enquanto a relação de igualdade pode ser utilizada tanto para valores inteiros quanto lógicos, observando que valores lógicos devem ser comparados a valores lógicos e valores inteiros devem ser comparados a valores inteiros. Na Figura 13, há um programa que exemplifica o uso dos operadores relacionais.

```
1   programa expressoes_relacionais
2
3       inteiro a b c
4
5   inicio
6
7       a <- 1
8       b <- 2
9       c <- 3
10
11      escreva a < b  //verdadeiro
12      escreva a = b // falso
13      escreva c > a // verdadeiro
14
15
16  fimprograma
```

Figura 13 - Exemplos de expressões relacionais

3.1.5.3 Expressões lógicas

As expressões lógicas são sentenças que após verificar a relação entre valores lógicos, através de operadores lógicos, produzem como resultado um valor lógico. Existem apenas dois valores lógicos o valor Verdadeiro e o valor Falso.

No caso das operações aritméticas, as constantes eram os números, já no caso das expressões lógicas, foram definidas as letras maiúsculas V e F para representar as constantes verdadeiro e falso.

A linguagem Simples disponibiliza três tipos de operações lógicas, a conjunção, a disjunção e a negação, sendo elas representadas pelos operadores “e”, “ou” e “nao”. Operações de conjunção disjunção são binárias e a operação de negação é unária.

3.1.5.4 Combinando diferentes tipos de operadores

Ao falar sobre o tipo de um operador, fala-se sobre o tipo do valor que será produzido pela operação.

Para produzir sentenças mais complexas, o programador pode definir expressões que combinam diferentes tipos de operadores, contanto que sejam respeitados os tipos das constantes e variáveis sobre as quais o operador está definido. A seguir (Figura 14), um exemplo:

```
1   programa combinacao_exp
2
3       inteiro a
4
5   inicio
6
7       leia a
8
9       se (a > 0) e (a < 10) entao
10          escreva a
11      senao
12          escreva 0
13      fimse
14
15  fimprograma
16
```

Figura 14 - Combinação de operadores lógicos e relacionais

No exemplo apresentado, há um programa que verifica se o valor informado pelo usuário é maior do que zero e menor do que dez e então escreve o valor, caso a condição se verdadeira, ou escreve zero, caso a condição seja falsa. Na condição do comando de desvio condicional existem dois operadores relacionais e um operador lógico. Ao encontrar a expressão, o programa resolverá, primeiramente, as duas expressões relacionais, que, como entrada, possuem variáveis e constantes do tipo inteiro e produzem como resultado um valor lógico. Depois de resolvidas, os resultados das expressões relacionais servirão de entrada para a expressão lógica, que receberá valores lógicos como entrada e produzirá um valor lógico como resultado. O resultado de toda a expressão será um valor lógico que será utilizado pelo comando de desvio condicional para decidir qual bloco de

comandos será executado. O mesmo raciocínio pode ser aplicado a expressões que utilizem outras combinações entre operadores de diferentes tipos.

3.2 A Máquina Virtual Simples

Definida a linguagem de programação Simples, o próximo passo foi criar um mecanismo que permitisse a execução dos programas escritos em Simples. Para tal, foi necessário traduzi-los para uma linguagem de máquina para, assim, serem executados pelas máquinas.

Partindo da necessidade de tornar o compilador independente de arquitetura e mais simples, a alternativa adotada pelo projeto foi a utilização de uma máquina virtual. A máquina virtual é um *software,* um simulador, que se comporta da mesma forma que uma máquina real se comportaria.

3.2.1 Definição da Máquina Virtual Simples (MVS)

A MVS foi inspirada na MEPA (Máquina para Execução Pascal) (KOWALTOWSKI, 1983). Assim como a MEPA, a MVS é uma máquina de pilha, característica muito apropriada para tradução de estruturas encontradas em linguagens de programação *C-Like* (derivadas da linguagem C) e *Pascal-Like* (derivadas da linguagem Pascal), como recursividade, estruturas aninhadas, etc.

A MVS possui uma memória divida em duas regiões: a região do programa (Vetor P) e a região da pilha de dados (Pilha M). O Vetor P é responsável por armazenar as instruções do programa, logo, o primeiro passo para a execução de um programa na MVS é carregar as instruções de máquina na memória. Enquanto o vetor P armazena as instruções a serem executadas, a pilha M é a região de memória que contem os valores que podem ser manipulados pelas instruções.

Além das duas regiões de memória citadas, a MVS possui três registradores especiais em sua arquitetura. Os registradores especiais receberam os nomes de “i”, “s” e “d”:

- O registrador “i” é um contador de programa, ou seja, armazena a posição do vetor P que contem a próxima instrução do programa a ser executa, denota P[i];
- O registrador “s” é a posição do elemento que está no topo da pilha M, sendo o valor do elemento descrito como M[s];
- O registrador “d” é utilizado para armazenar uma posição da pilha M. Sendo útil para os casos de mudança de escopo dentro de um programa.

O funcionamento da MVS consiste em carregar as instruções no vetor P e inicializar os registradores para que, em seguida, as instruções apontadas pelo registrador “i” sejam executadas até que seja encontrada uma instrução para parar a máquina ou um erro ocorra. Após a execução de uma operação, o contador “i“ passa a apontar para a próxima instrução do vetor P, exceto nos casos em que a instrução executada altera o registrador.

3.2.2 A linguagem da Máquina Virtual Simples

No projeto da MVS, além da arquitetura da máquina, foi necessário definir, também, a linguagem reconhecida pela máquina, ou seja, o conjunto de instruções que a máquina reconhece.

Uma vez que as instruções de máquina foram desenvolvidas para serem entendidas por máquinas, espera-se que estas sejam bem distantes da linguagem natural do ser humano, podendo, então ser de difícil entendimento. A fim de manter a característica de ser uma linguagem bastante reduzida, as instruções da MVS foram definidas como sendo palavras de quatro letras. Porém, apesar de bastante reduzida, a linguagem manteve uma certa legibilidade como consequência da escolha de instruções mnemônicas.

No projeto da linguagem da MVS foram propostas vinte e uma instruções, suficientes para dar suporte a todas as funcionalidades oferecidas pela linguagem de programação Simples. Posteriormente, foram acrescentados mais dez comandos para dar suporte às operações de mudança de escopo, manipulação de variáveis locais e demais funcionalidades necessárias para a implementação de programas na linguagem Simples que utilizem subprogramas. Na Tabela 1, são apresentadas as

trinta e uma instruções da MVS, além da sentença a qual o código mnemônico faz referência.

Tabela 1 - Instruções da MVS

#	Instrução	Descrição
1	INPP	Inicia programa principal
2	FIMP	Fim do programa
3	AMEM k	Aloca Memória
4	CRCT k	Carrega constante
5	CRVG n	Carrega variável global
6	ARZG n	Armazena em variável global
7	DSVS p	Desvia sempre
8	DSVF p	Desvia se falso
9	LEIA	Leitura
10	ESCR	Escrita
11	CMMA	Compara se maior
12	CMME	Compara se menor
13	CMIG	Compara se igual
14	DISJ	Disjunção
15	CONJ	Conjunção
16	NEGA	Negação
17	SOMA	Soma
18	SUBT	Subtração
19	MULT	Multiplicação
20	DIVI	Divisão
21	NADA	Não faz nada
22	CRVL n	Carrega variável local
23	ARZL n	Armazena variável local
24	CREL n	Carrega endereço local
25	CREG n	Carrega endereço global
26	CRVI n	Carrega valor indireto
27	ARMI n	Armazena indireto
28	SVCP	Salva contador de programa
29	ENSP	Entrada subprograma
30	RTSP n	Retorno subprograma
31	DMEM n	Desaloca memória

Na Tabela 1, observando as instruções, pode-se ver que algumas delas estão acompanhadas de uma letra. A letra significa que a instrução possui um operando. Por exemplo, na instrução AMEM, temos uma letra n que diz quantas posições de memória serão alocadas na pilha. Além de um operando, uma instrução pode possuir, também, um rótulo, utilizado para fazer referência a uma instrução, facilitando nas operações de desvio.

3.3 O Compilador Simples

Após definir a linguagem de programação Simples e a máquina para a qual geraremos código, o próximo passo é a construção do compilador. Neste trabalho não será apresentada a implementação detalhada do projeto do Compilador Simples, serão apresentadas apenas suas funcionalidades e as ideias utilizadas no desenvolvimento.

O Compilador Simples, além da função de traduzir código Simples para código da MVS, também é uma ferramenta didática. A construção do compilador foi feita de forma a privilegiar o entendimento dos alunos que o utilizarem.

Questões como desempenho, otimização de código e consumo de memória, que normalmente seriam de extrema importância para um compilador utilizado comercialmente, não foram o foco aqui.

O objetivo é que os alunos da disciplina de compiladores, com o auxílio do professor, após aprender toda a fundamentação teórica da disciplina, sejam capazes de olhar para o projeto do Compilador Simples e entender o seu funcionamento. Além de entender, os alunos poderão desenvolver novas funcionalidades.

O professor poderia solicitar aos discentes que acrescentassem um novo comando na linguagem Simples, ou, ainda, poderia retirar algum tratamento semântico já implementado para que os alunos, vendo o compilador funcionar, procurem soluções e implementem o tratamento de modo que seus compiladores funcionem da mesma forma que o compilador fornecido pelo professor.

3.3.1 Desenvolvimento do Compilador Simples

Para a construção do compilador foi necessário o desenvolvimento três fases fundamentais: a análise léxica, a análise sintática e a análise semântica.

O Compilador Simples realiza uma compilação dirigida à sintaxe, o que significa que à medida que o analisador sintático é executado e as construções vão sendo identificadas, são tomadas ações semânticas.

Para desenvolver o analisador sintático, foi utilizado o JCup, uma ferramenta que, a partir da especificação da gramática da linguagem cria o código Java do analisador sintático. O JCup possui integração nativa com o JFlex. Enquanto o JCup é uma ferramenta utilizada para gerar o analisador sintático, o JFlex é uma ferramenta utilizada para gerar o código Java de um analisador léxico. O JCup e o JFlex são baseados no YACC e no FLEX do UNIX.

Do ponto de vista da teoria da computação, a análise léxica está relacionada às linguagens regulares, que podem ser reconhecidas por autômatos finitos determinísticos, a análise sintática está associada às linguagens livres do contexto, que podem ser resolvidas por autômatos de pilha, em ambos os casos os autômatos são relativamente simples de serem implementados e, talvez, esse seja o motivo para a existência de ferramentas como o JCup e o JFlex.

A análise semântica está relacionada às linguagens sensíveis ao contexto, que podem ser reconhecidas por uma Máquina de Turing, muito mais complexa do que os autômatos citados anteriormente. Por este motivo, para validar se um programa Simples está semanticamente correto, ao invés de tentar verificar se o programa pertence a uma gramática sensível ao contexto, a alternativa mais simples é a utilização de estruturas auxiliares para verificar as regras.

O analisador semântico implementado pelo compilador Simples funciona através da manutenção da tabela de símbolos e de uma pilha semântica. A tabela de símbolos tem a função de armazenar as informações a respeito dos símbolos encontrados, tais como, identificadores, tipo, classificação, etc. Já a pilha semântica tem função de armazenar resultados de operações semânticas para que sejam utilizados futuramente. Como exemplo do uso da tabela de símbolos, tem-se a verificação do uso de variáveis, pois as variáveis são inseridas na tabela na seção de declaração de variáveis, caso o compilador encontre uma variável, no corpo do programa, e ela não estiver na tabela de símbolos, significa que ela não foi

declarada, caracterizando, assim, um erro semântico. Já um exemplo de utilização da pilha semântica pode ser visto no tratamento de tipos. Em uma operação de aritmética, por exemplo, deve-se utilizar apenas variáveis ou constantes do tipo inteiro, então, à medida que os valores são carregados para a memória, seus tipos são armazenados na pilha, para que no momento da operação seja verificado se os tipos do operador e dos operandos são compatíveis.

4
Resultados

Neste capítulo serão explanados os resultados alcançados com o desenvolvimento da ferramenta descrita neste trabalho.

4.1 O Ambiente de Desenvolvimento Integrado Simples e a Máquina Virtual Simples

Como resultado, foi produzida uma ferramenta didática para o ensino de lógica de programação, compiladores e arquitetura de computadores.

A ferramenta é composta pelo Compilador Simples, pelo simulador da MVS e por editor de código fonte Simples. O editor foi chamado de Ambiente de Desenvolvimento Integrado Simples, pois a partir da interface do editor, o usuário pode compilar, executar e depurar programas Simples. O ambiente possui ainda uma integração com o simulador da MVS, como será mostrado mais a diante.

4.1.1 Interface principal do utilizador

Ao iniciar o sistema, o usuário tem acesso à interface principal (Figura 15). Com exceção das funcionalidades que envolvem o simulador da MVS, a tela principal é utilizada para a execução de todas as funções implementadas.

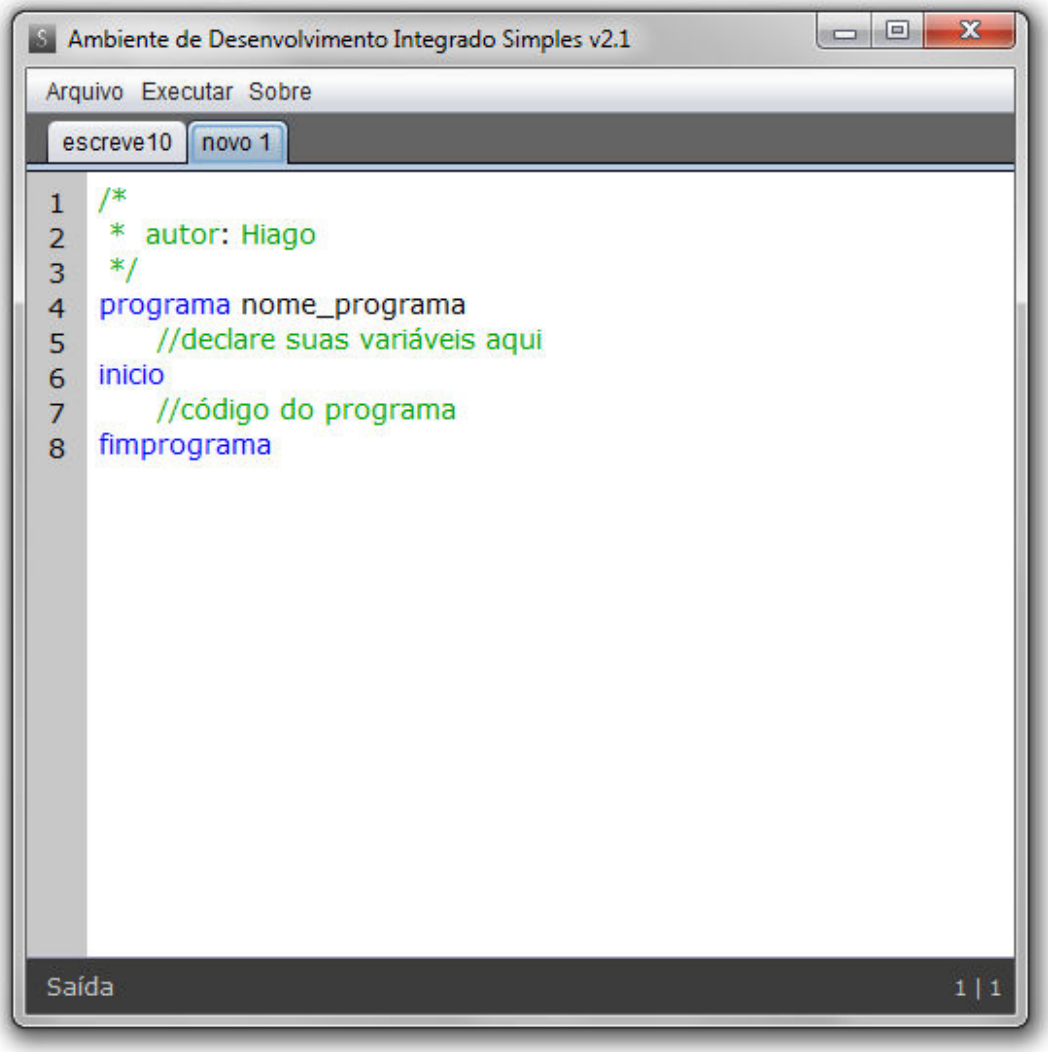


Figura 15 - Tela principal do sistema

A tela principal do sistema tem, ocupando a maior área, o editor de código fonte Simples. O editor, como forma de auxílio ao programador, destaca diferentes tipos de palavras dentro do programa.

Os comentários, que serão ignorados pelo compilador, possuem a cor verde, as palavras utilizadas na definição de comandos e do corpo do programa, possuem a cor azul, e, como poderá ser observado em programas exibidos mais a adiante, as palavras destinadas aos tipos das variáveis, possuem a cor roxa. O sistema de cores, depois de implementado, trouxe um aspecto visual agradável ao editor. Acredita-se que com as cores será mais fácil para o programador ler um código, além de auxiliá-lo a lembrar quais as palavras definidas na linguagem, pois, caso se engane e utilize uma palavra diferente, poderá ver que essa não mudou de cor.

No topo da área de edição há um sistema de abas que permite ao desenvolver manter vários programas abertos, além de poder alternar entre eles. O nome presente na aba é equivalente ao nome do arquivo em que o programa está salvo. Para indicar que um programa não ainda não foi salvo desde a sua última edição é acrescentado um asterisco na frente do nome.

Além da área destinada à edição de programas, a tela principal do sistema tem duas áreas: uma barra de menus e um rodapé.

Um dos componentes presentes no rodapé está vinculado ao editor de código. Do lado direito do rodapé existem dois números separados por uma barra que representam a linha e coluna em que o cursor se encontra dentro do código fonte. Essa funcionalidade foi pensada com o objetivo de auxiliar o programador a encontrar o ponto do programa em que ocorreu um erro (os tipos de erros serão vistos mais a frente).

4.1.2 Menus

Os menus, presentes na barra superior do sistema, dão acesso a todas as funcionalidades desenvolvidas. Os menus foram separados em três categorias: Arquivo, Executar e Sobre.

4.1.2.1 Menu Arquivo

O menu Arquivo traz a funções necessárias para a interação do sistema com os arquivos onde os programas Simples estão salvos. Na figura são apresentadas as funções presentes no referido menu:

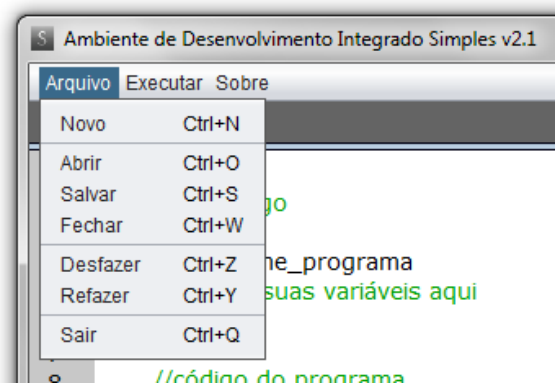


Figura 16 - Menu Arquivo

As funções do menu Arquivo são:

- Novo: cria um programa Simples vazio, com apenas a definição do programa e a indicação das seções de declaração de variáveis e do corpo do programa.
- Abrir: exibe uma janela para que o usuário possa navegar nos arquivos do seu computador, podendo escolher um programa Simples para carregá-lo no editor.
- Salvar: salva o programa Simples presente no editor em um arquivo. Na primeira vez que esta opção é utilizada em um programa recém-criado, o sistema pedirá ao usuário para escolher a localização e o nome do nome arquivo.
- Fechar: fecha o programa que está em edição. Caso o programa não tenha sido salvo ainda, uma mensagem será exibida.
- Desfazer: elimina a última alteração feita no programa em edição.
- Refazer: restaura uma alteração do programa que foi eliminada com o comando de desfazer.
- Sair: finaliza o uso da ferramenta fechando o programa. Caso exista algum programa ainda não salvo, o sistema exibirá uma mensagem.

Todas as funções do menu Arquivo possuem um atalho associado, ou seja, é possível utilizar uma função pressionando uma combinação de teclas. As teclas de atalho para cada menu podem ser vistas na figura do menu Arquivo.

4.1.2.2 Menu Executar

O menu Executar agrega as principais funcionalidades do sistema, sendo elas: Executar Programa, Depurador, Compilador, Executar Máquina Virtual Simples. De forma similar ao menu Arquivo, todas as funções do menu Executar possuem um atalho do teclado. Mais a frente, cada um das funcionalidades do menu Executar será melhor detalhada.

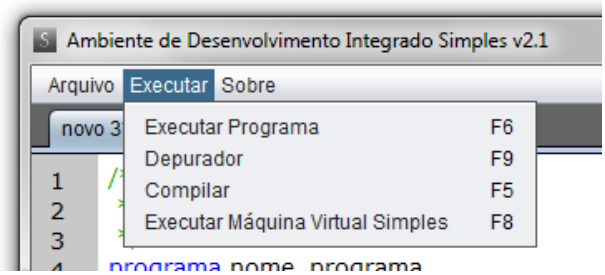


Figura 17 - Menu Executar

4.1.2.3 Menu Sobre

O menu Sobre possui apenas a função de exibir informações sobre o projeto, como, por exemplo, nome e contato dos autores.

4.1.3 O Simulador da Máquina Virtual Simples

Além de implementar a MVS, que tem a função de executar o código gerado pelo Compilador Simples, foi desenvolvida uma representação gráfica da MVS, que além de exibir o estado interno da máquina, permite que a execução da máquina seja controlada pela usuário.

O simulador pode ser utilizado em dois cenários: de forma integrada ao editor de código Simples ou de forma isolada do restante do projeto. Ter um *software* composto apenas pelo simulador da MVS possibilita ao professor de compiladores elaborar atividades em que os alunos façam o trabalho do Compilador Simples, traduzindo os programas à mão para, depois, submeter o código de máquina, que traduziram, ao simulador para avaliar a corretude da tradução. Talvez, traduzir alguns programas bastante trivias à mão, antes de conhecer a implementação do Compilador Simples, facilite no entendimento dos alunos.

A utilização do simulador é bem intuitiva. Uma vez carregadas as instruções, o usuário precisa, apenas, utilizar o botão que faz que com que a próxima instrução seja executada. O simulador executará as instruções, uma a uma, conforme o usuário apertar o botão. Desta forma, o aluno poderá utilizar o simulador no ritmo que julgar ideal, possibilitando um aprendizado mais efetivo.

Na figura a seguir, pode-se ver uma imagem do simulador:

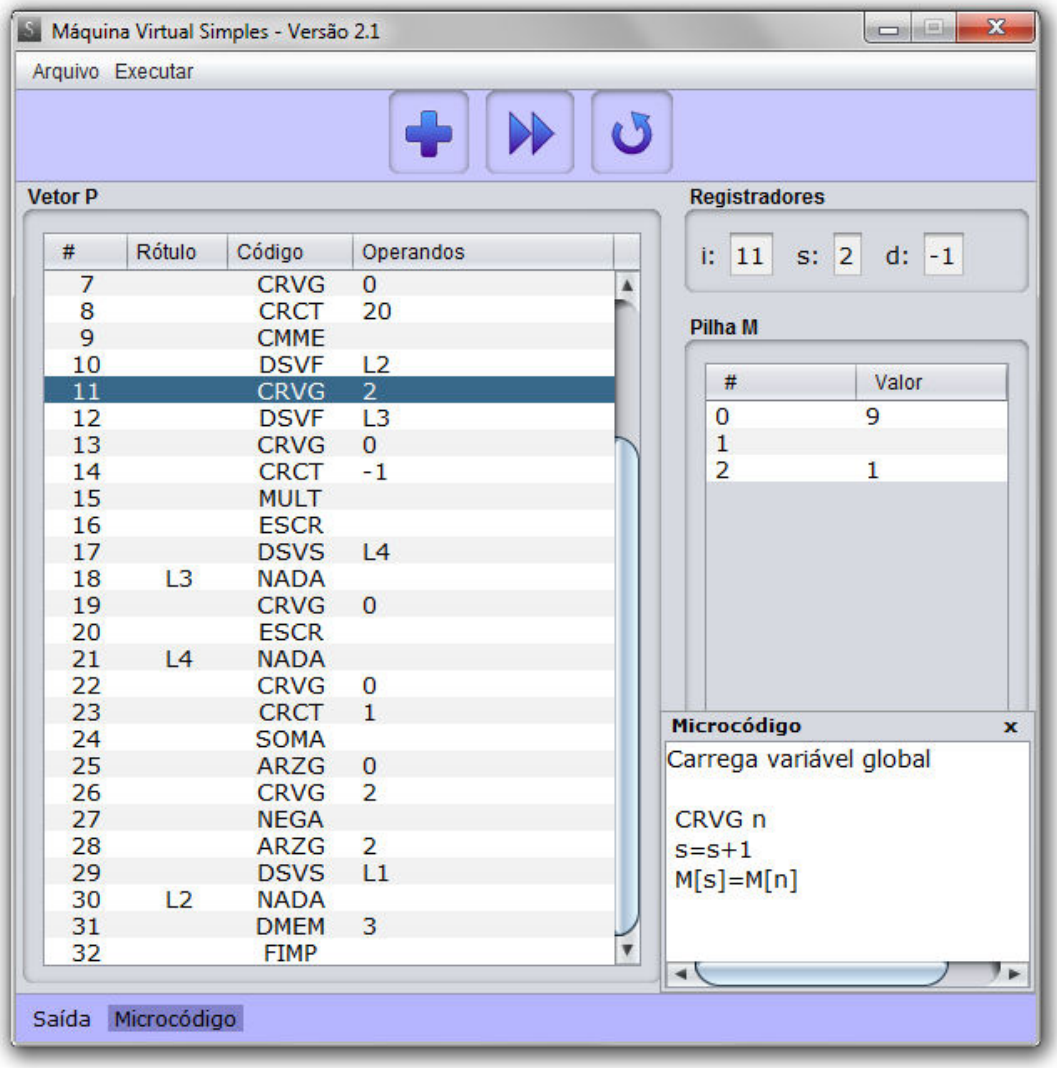


Figura 18 - Simulador da MVS

A interface gráfica do simulador traz uma representação do estado interno da máquina através da exibição do Vetor P, da Pilha M e dos registradores "i", "s" e "d".

Para tornar a simulação mais didática, criou-se um quadro com o título de microcódigo. Sua função é trazer uma pequena descrição da próxima instrução a ser executada, além de um pequeno trecho de pseudocódigo que detalha a ação da instrução sobre o estado da máquina. A exibição deste quadro é opcional, podendo o usuário decidir se ele será exibido ou não. Para alternar o estado de exibição do quadro, basta clicar sobre a palavra "microcódigo", presente no rodapé da tela.

O simulador possui apenas dois menus: Arquivo e Executar. O menu Arquivo possui as funcionalidades de carregar um programa a partir de um arquivo e de salvar o programa carregado em um arquivo a se escolher. O menu executar possui três funções: carregar um novo programa, executar a próxima instrução do programa em execução e reiniciar a execução do programa carregado. Essas três opções podem, também, ser utilizadas através dos botões presentes no topo da tela.

Para facilitar a memorização de qual é a função de cada um dos botões, foram utilizados ícones semelhantes aos de um *player*, onde a seta faz com que a próxima música seja tocada, o "mais" é utilizado para adicionar músicas e a seta circular repete uma música.

Para finalizar a explicação dos componentes do simulador, descreve-se as operações de entrada e saída. Quando a instrução LEIA é executada pela MVS, é necessário que o usuário informe um valor, que será carregado na Pilha M. Neste caso, ao executar a instrução, o Simulador exibe uma janela responsável por ler o valor informado pelo usuário. Ao executar a operação ESCR, o simulador deve exibir um valor. Para este fim, existe um painel chamado saída, que exibe o resultado da execução do programa. Sempre que necessário, o painel é exibido de forma automática. Para visualizar o painel a qualquer momento, o usuário pode clicar sobre a palavra "saída", presente no rodapé da tela.

4.1.4 Funcionalidades da ferramenta

Conforme o proposto pela ferramenta, suas principais funcionalidades são: compilar, executar e depurar programas Simples e mostrar o funcionamento da MVS. Cada uma das funcionalidades será apresentada nas seções seguintes:

4.1.4.1 Utilização do Compilador Simples

A tarefa de compilar programas Simples é acionada por meio do Ambiente de Desenvolvimento Integrado Simples, basta que o usuário acione a função compilar e o programa que estiver aberto no editor será compilado. Estando o programa correto, a compilação será realizada e o código de máquina produzido será exibido na saída do ambiente, como mostrado na Figura 19.

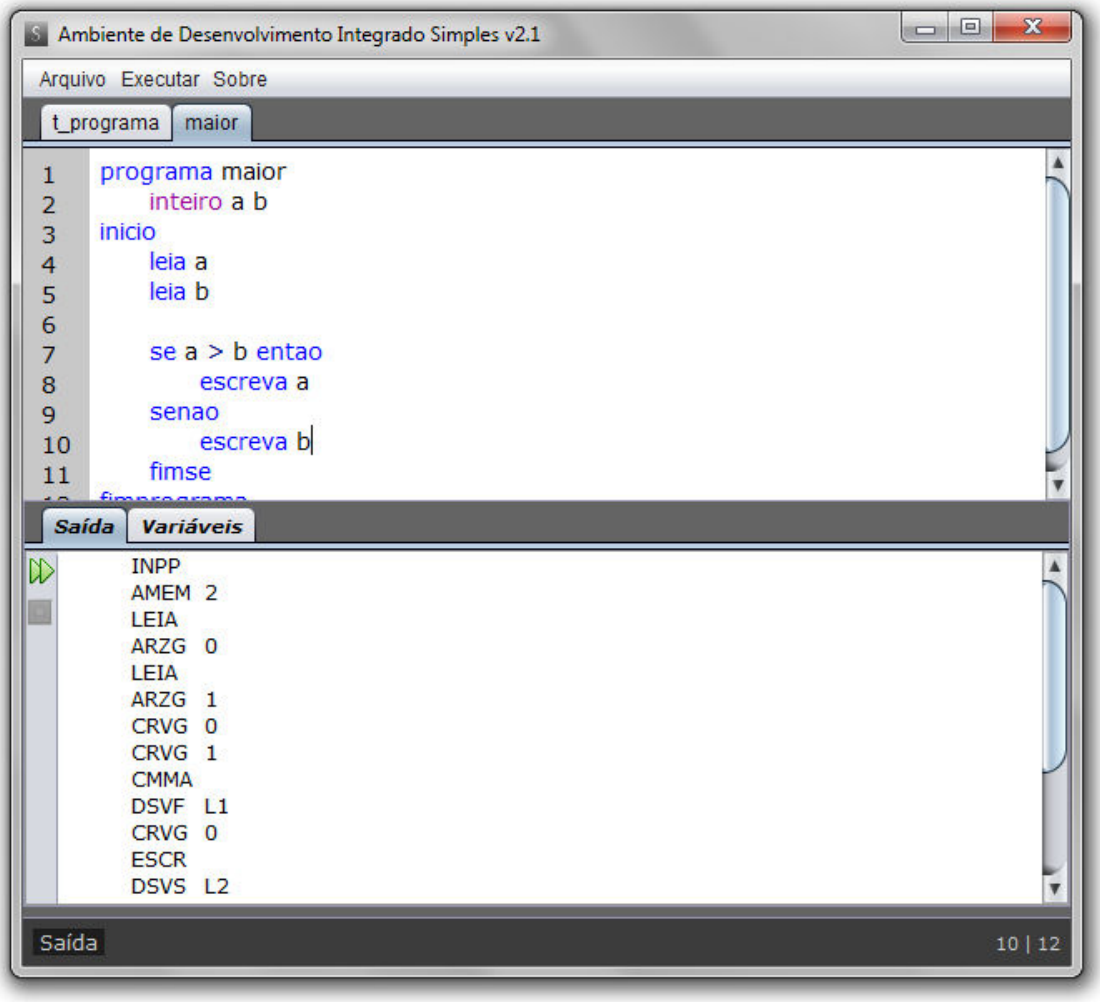


Figura 19 - Resultado da compilação de um programa

Nos casos em que o programa a ser compilado estiver incorreto, seja por um erro léxico, sintático ou semântico, uma notificação de erro será exibida. Quando possível, o ponto em que se encontra o erro é exibido. Para exemplificar as situações de erros, serão exibidas três figuras, contendo um erro léxico, um erro sintático e um erro semântico respectivamente.

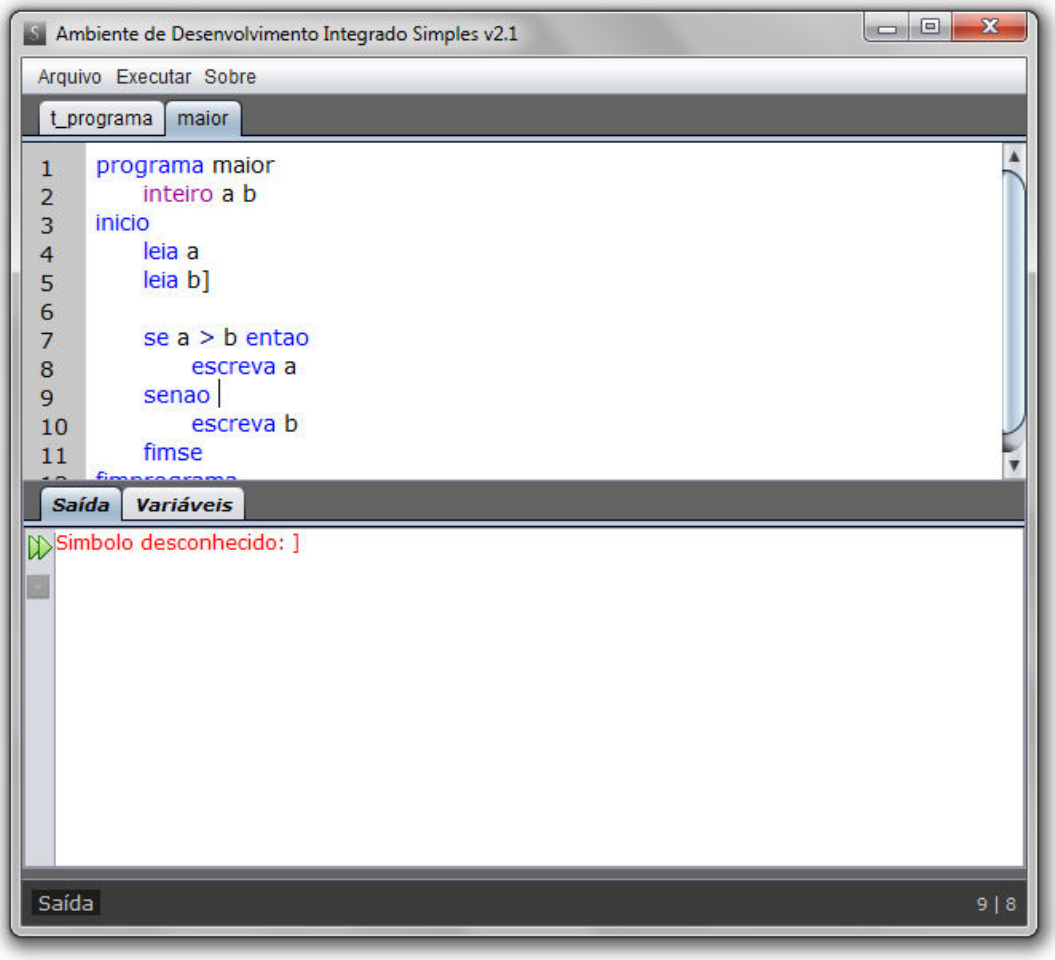


Figura 20 – Exemplo de erro léxico

Um erro léxico é caracterizado pela presença de um elemento que não é reconhecido pelo analisador léxico. Nestes casos, o símbolo não pertencente à linguagem é exibido.

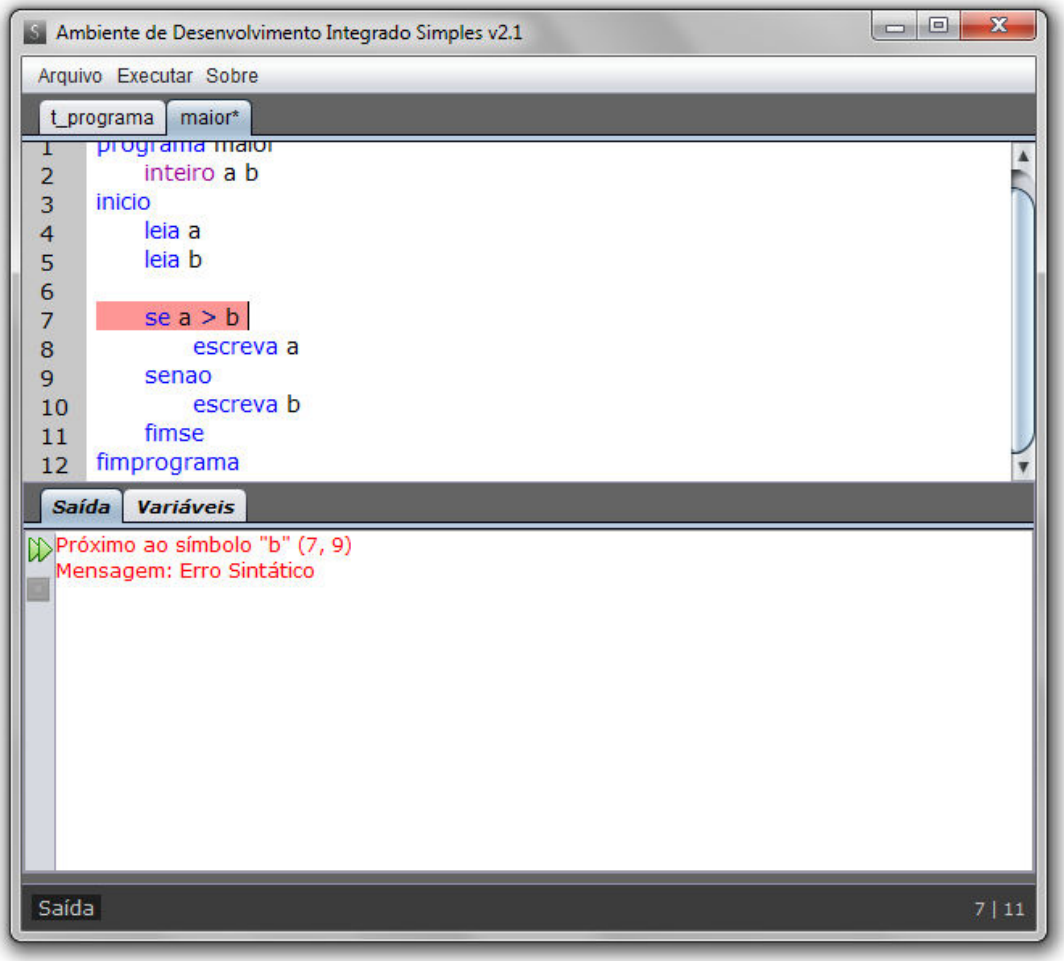


Figura 21 - Exemplo de erro sintático

Durante a análise sintática do programa, é verificado se todas as construções são validadas. No exemplo apresentado, a construção do comando de desvio condicional não está correta, pois, o programador se esqueceu da palavra “entao”, que deve aparecer após o teste lógico. Em casos de erro sintático, o compilador informa o ponto que precede o erro. Para que o programador encontre o ponto do erro mais facilmente, o ambiente destaca a linha que contêm o erro.

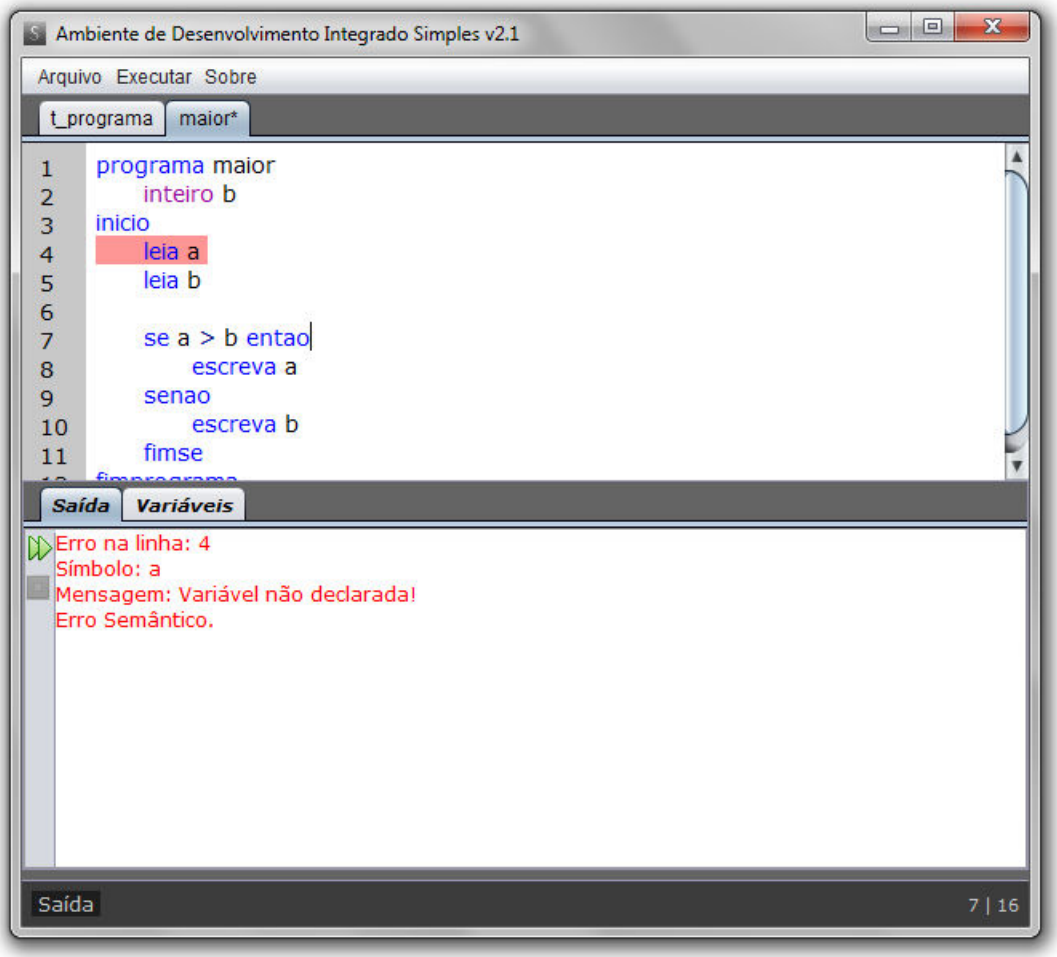


Figura 22 - Exemplo de erro semântico

Quando um programa não pode ser compilado devido a um erro semântico, ou seja, o programa está sintaticamente correto, mas algo relacionado ao contexto do programa está errado, o compilador finaliza a compilação com uma mensagem de erro que contém o local onde o erro ocorreu e uma descrição da causa do erro. Neste caso, o ambiente, novamente, destaca a linha que contém o erro.

É importe ressaltar que o compilador, ao encontrar um erro, aborta a compilação imediatamente. Caso existam vários erros no programa, apenas o primeiro erro encontrado pelo compilador será apresentado.

4.1.4.2 Execução de programas

Para ver seus programas em funcionamento, o desenvolver de acionar a função de "Executar Programa". Nesta opção o programa é compilado e submetido a uma

MVS embutida no ambiente. Uma vez que a interface gráfica do simulador da MVS não é exibida e todas as instruções são executadas em sequência, a utilização da máquina se torna transparente ao desenvolvedor. A execução do programa ocorre de forma ininterrupta, até que o programa termine, com exceção dos casos onde é necessário que o usuário informe um valor solicitado por um programa de leitura.

No caso dos comandos de escrita, os valores são exibidos em uma área destinada à saída do programa. A seguir, é apresentado um exemplo em que um programa que tem a função de escrever os números de zero a nove foi executado:

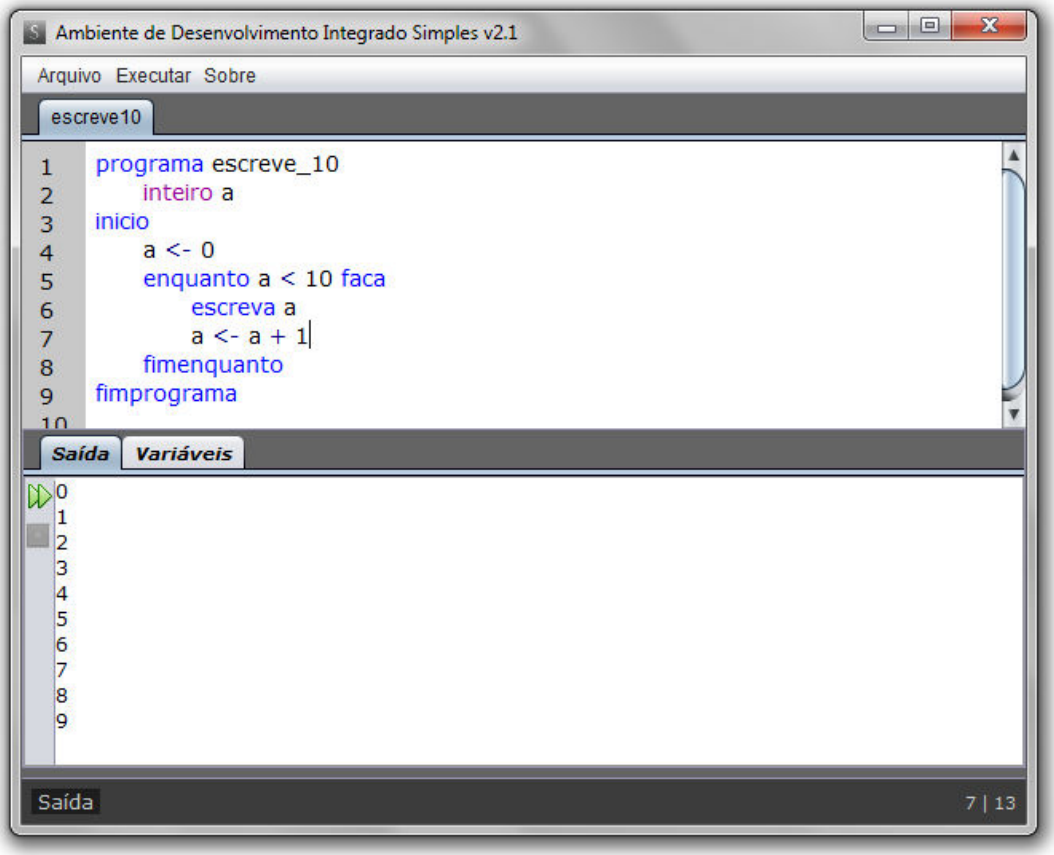


Figura 23 - Execução de um programa

4.1.4.3 Depuração de programas

A depuração de um programa é bastante similar à execução, possuindo apenas uma diferença: na depuração, o programa é executado linha a linha, sendo o avanço das linhas controlado pelo usuário.

No modo de depuração, o ambiente destaca a próxima linha a ser executada e exibe, de forma dinâmica, o valor das variáveis em um dado instante. Ao entrar no modo de depuração, o ambiente exibe dois novos botões, sendo um utilizado para avançar as linhas e o outro para encerrar o modo de depuração.

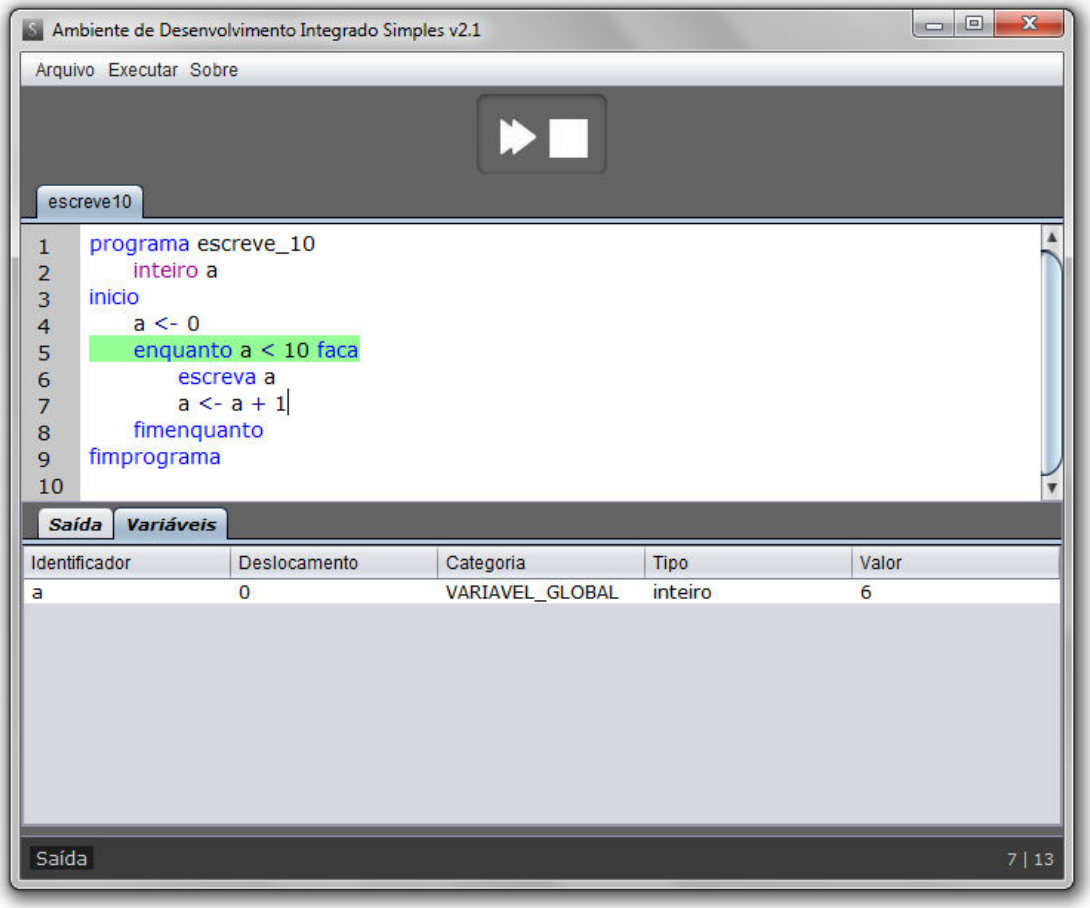


Figura 24 - Programa em modo de depuração

Executar um programa linha a linha pode ser uma boa maneira de entender o funcionamento de um programa que não apresentou o comportamento esperado. Esta funcionalidade também pode ser muito útil ao professor de algoritmos, que pode utilizar a ferramenta para explicar a lógica de um programa aos alunos.

4.1.4.4 Execução da Máquina Virtual Simples

Em uma seção anterior, foi explicado o funcionamento do simulador da MVS, nesta seção o foco será a integração entre o ambiente de desenvolvimento e o simulador.

Ao selecionar a opção “Executar Máquina Virtual Simples”, o código Simples presente no editor será compilado e enviado ao simulador da MVS, que passará a ser exibido. O simulador e o ambiente de desenvolvimento ficarão lado a lado na tela. A partir deste ponto, o simulador pode ser utilizado normalmente, apresentando o mesmo comportamento já descrito. Enquanto o simulador é executado, o ambiente exibe a tabela de símbolos gerada no processo de compilação, o valor atual das variáveis presentes na tabela de símbolos e destaca a linha do programa Simples que deu origem à instrução que está em execução na MVS.

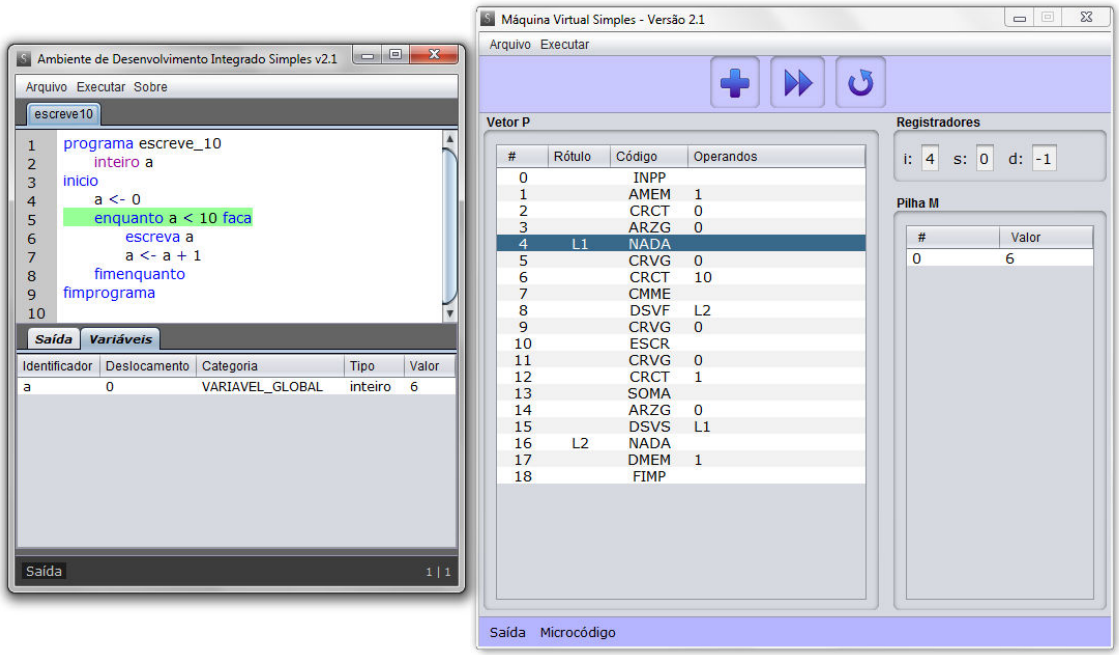


Figura 25 - Execução integrada da MVS

Acredita-se que, com esta funcionalidade, os alunos poderão entender, de forma clara, a relação entre uma linguagem de programação e uma linguagem de máquina e a relação entre a Pilha M e as variáveis utilizadas no programa. Talvez esta seja a maior contribuição da ferramenta: executar as instruções da MVS, uma a uma, observando o estado da máquina, ao mesmo tempo em que o programa Simples avança linha a linha, exibindo o valor de suas variáveis, pode ser um recurso didático poderoso, que permitirá ao docente evidenciar a relação entre a linguagem de programação, o compilador e a arquitetura da máquina.

4.2 Aceitação da ferramenta

Objetivando avaliar a aceitação da ferramenta pelos alunos que a utilizaram na disciplina de compiladores do sexto período do curso de ciência da computação da Unifal-MG, foi realizada uma pesquisa informal, através de um questionário eletrônico (Apêndice I), composto por seis perguntas que, basicamente, avaliam a experiência de utilização da ferramenta, o potencial da ferramenta para o ensino de algoritmos, o quanto cada funcionalidade implementada auxiliou no aprendizado de compiladores e se a ferramenta atingiu os objetivos propostos.

Antes da utilização a ferramenta, o professor forneceu um material teórico aos alunos que descrevia cada um dos componentes, além de alguns exemplos de utilização.

O questionário foi respondido por dezessete alunos e o resultado se mostrou bastante satisfatório. Em relação à linguagem de programação Simples, todos os alunos, no mínimo, consideram que se saíram bem ao desenvolver programas pela primeira vez e, com exceção de dois alunos, acreditam que a utilização da ferramenta para o ensino de algoritmos seria interessante.

Quando avaliadas as funcionalidades utilizadas para o ensino de compiladores, a possibilidade de exibir a tabela de símbolos durante os modos de depuração foi avaliada, por um aluno, como sendo indiferente. Fora a avaliação citada, todas as funcionalidades foram consideradas boas ou ótimas, sendo a alternativa ótima a mais frequente.

Em uma escala de um (pior) a dez (melhor), a ferramenta, quando avaliada a utilização na disciplina de compiladores, recebeu onze vezes a nota dez, atingindo uma nota média de 9.35.

Na questão que avalia a contribuição da ferramenta para o entendimento de arquiteturas de compiladores, obteve-se que 88% dos alunos, após a utilização do recurso didático, passaram a compreender melhor a relação entre as instruções executadas pela máquina a memória e os registradores.

Para encerrar o questionário, os alunos deveriam responder se, a partir da ferramenta proposta, seriam capazes de explicar a relação entre a definição de uma linguagem de programação, um compilador e a arquitetura da máquina utilizada.

Nesta questão 100% dos alunos responderam de forma positiva. Sendo este ponto um dos principais objetivos do projeto, pode-se dizer que, o resultado obtido é um forte indício de que o projeto atingiu o objetivo proposto.

O resultado completo do questionário realizado, contendo o número de vezes que cada alternativa foi escolhida, as respectivas porcentagens e representações gráficas, pode ser visto no apêndice II.

5
Conclusões

Este capítulo apresenta as conclusões obtidas a partir desta monografia. Além de algumas sugestões para trabalhos futuros.

Pode-se afirmar que a construção da ferramenta proposta é viável, uma vez que, esta foi implementa e colocada em teste.

Embora não se possa afirmar, por falta de mais estudos, que a ferramenta é um recurso didático ideal para o ensino de compiladores, há fortes indícios de que sua utilização traz grandes benefícios à aprendizagem.

Hoje, a utilização da ferramenta já é uma realidade dentro do curso de ciência da computação da Unifal-MG, estando presente na disciplina de compiladores.

5.1 Trabalhos Futuros

A fim de aumentar o tempo de utilização da linguagem Simples, que precede a migração para uma linguagem de programação mais robusta, no ensino de algoritmos, poderiam ser implementadas novas funcionalidades, como, por exemplo, a utilização de vetores e outros tipos de variáveis. Desta forma, um número maior de conceitos poderiam ser abordados pelo docente, ao utilizar a ferramenta.

Visando a difusão do recurso didático desenvolvido, seria interessante a realização de estudos sobre a utilização da ferramenta nas disciplinas de algoritmos e arquitetura de computadores da Unifal-MG, além de outras universidades.

6
Referências Bibliográficas

AHO, A. V.; LAM, M. S.; SETHI, R.; ULLMAN, J. D. Compiladores: princípios, técnicas e ferramentas. São Paulo: Pearson Education do Brasil, 2008.

ALMEIDA, E. S.; COSTA, E. B.; SILVA, K. S.; PAES, R. B.; ALMEIDA, A. A. M.; BRAGA, J. D. H. AMBAP: Um Ambiente de Apoio ao Aprendizado de Programação, Iniciação Científica, Workshop de Educação em Computação, Congresso Anual da SBC, Florianópolis, 2002

DEBRAY, S. Making compiler design relevant for students who will (most likely) never design compiler. Proceedings of the 33th SIGCSE technical symposium on Computer science education, Covington, Kentucky, USA, 2002

EREMIN, E. Educational Model of Computer as a Base For Informatics Learning. International Journal "Information Theories & Applications" Vol.12, 2005.

HENRY, T. R. Teaching compiler construction using a domain specific language. Proceedings of the 36th SIGCSE technical symposium on Computer science education, St. Louis, Missouri, USA, 2005.

KOWALTOWSKI, T. Implementação de Linguagens de Programação. Rio de Janeiro: Guanabara Dois S.A, 1983

ORUÇ, A. Y; GUNDUZHAN, E: A Visual Instruction Set Architecture and Simulation Tool for Computer Engineering Education. 33th ASEE/IEEE Frontiers in Education Conference, USA, 2003.

PRICE, A. M. A.; TOSCANI, S. S. Implementação de Linguagens de Programação: Compiladores. Instituto de Informática da UFRGS. Porto Alegre: Sagra, 2001

SCHNEIDER, C. S.; PASSERINO L. M.; OLIVEIRA R. F. Compilador Educativo Verto: ambiente de aprendizagem de compiladores VERTO Educacional Compiler: environment for compiler learning, 2005.

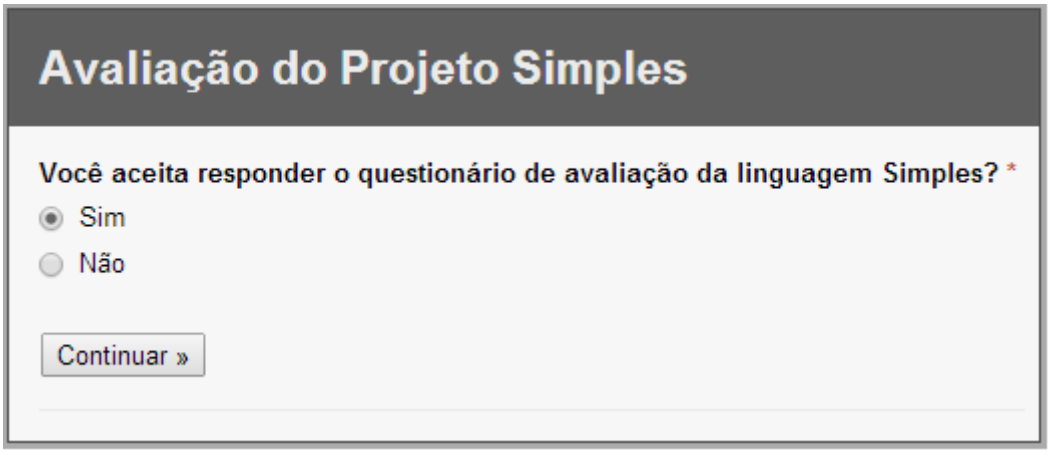
STALLINGS, W. Arquitetura e Organização de Computadores. 5.ed. Prentice-Hall Brasil, 2003

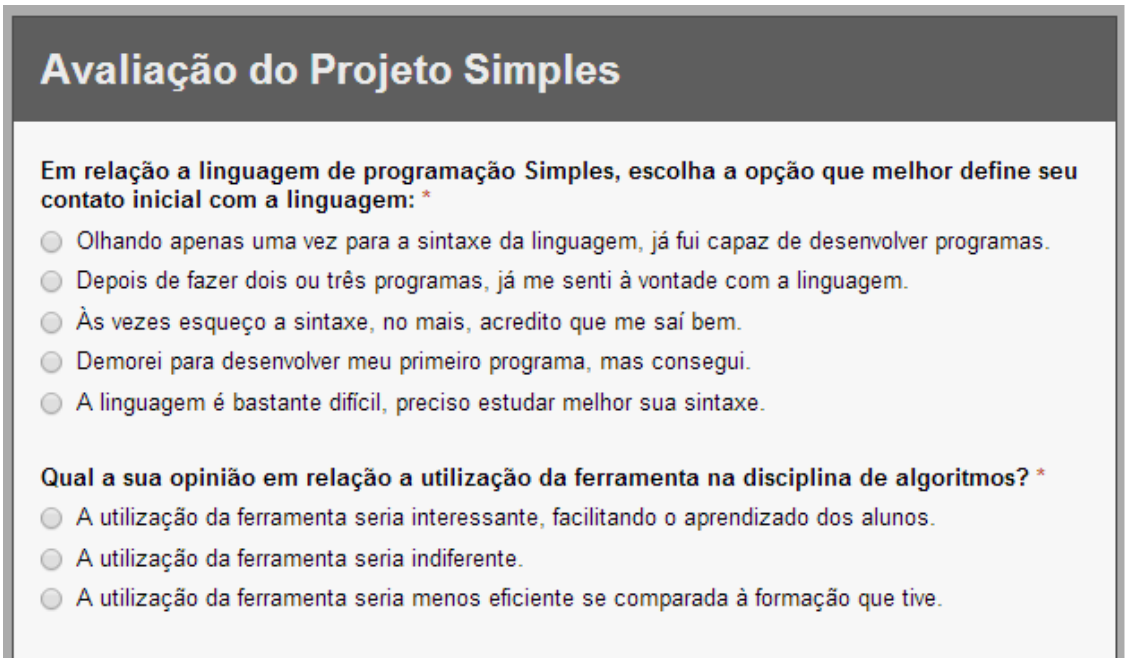
VIEIRA, P. V.; RAABE, A. L. A.; ZEFERINO, C. A.. Bipide Ambiente de Desenvolvimento Integrado para a Arquitetura dos Processadores BIP. Revista Brasileira de Informática na Educação, v. 18, p. 32-43, 2010.

7
Apêndices e Anexos

7.1 Apêndice I

Questionário eletrônico enviado aos alunos.





Como você avalia a contribuição de cada uma das funcionalidades da ferramenta para o ensino da disciplina de compiladores? *

	péssima	ruim	indiferente	boa	ótima
Executar a máquina passo a passo exibindo seu estado interno	○	○	○	○	○
Exibição da tabela de símbolos durante modos de depuração	○	○	○	○	○
Exibição da descrição dos microcódigos das instruções	○	○	○	○	○
Realce da linha de código fonte correspondente a instrução de máquina em execução	○	○	○	○	○

Em uma escala, onde um é a pior nota e dez, a melhor. Como você classificaria a utilização da ferramenta na disciplina de compiladores? *

1	2	3	4	5	6	7	8	9	10
○	○	○	○	○	○	○	○	○	○

Depois de utilizar o simulador da Máquina Virtual Simples, como você avalia o seu entendimento sobre o funcionamento interno de um computador? *

- ○ Mantenho a mesma visão que possuía.
- ○ Pude compreender melhor a relação entre registradores, memória e instruções de máquina.
- ○ Continuo não entendendo como um programa é executado por uma máquina.

A partir da utilização da linguagem, da máquina e do compilador, propostos, você, graduando em ciência da computação, seria capaz de responder qual a relação entre a definição de uma linguagem de programação, a arquitetura da máquina utilizada e o compilador? *

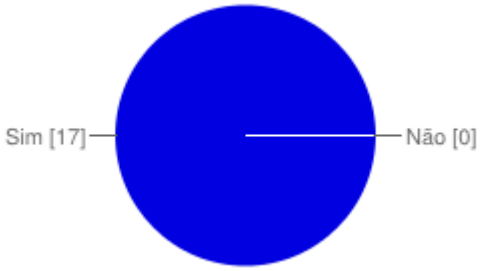
- ○ Sim
- ○ Não

« Voltar | Enviar

7.2 Apêndice II

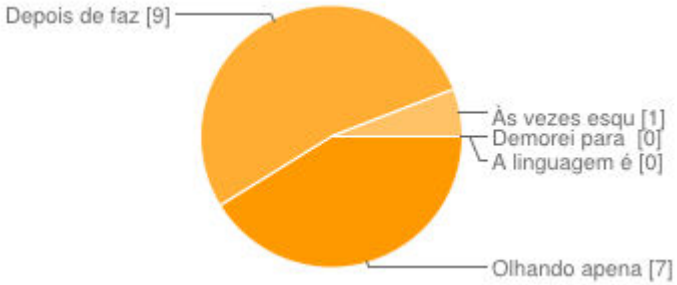
Resultado do preenchimento do questionário.

Você aceita responder o questionário de avaliação da linguagem Simples?



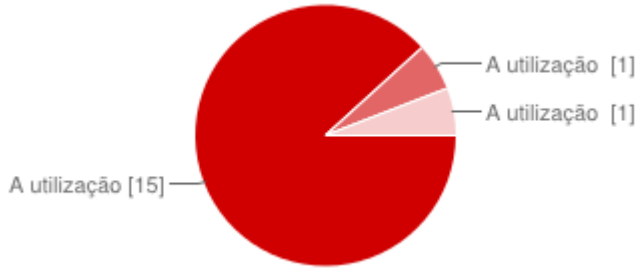
Sim	17	100%
Não	0	0%

Em relação à linguagem de programação Simples, escolha a opção que melhor define seu contato inicial com a linguagem:



Olhando apenas uma vez para a sintaxe da linguagem, já fui capaz de desenvolver programas.	7	41%
Depois de fazer dois ou três programas, já me senti à vontade com a linguagem.	9	53%
Às vezes esqueço a sintaxe, no mais, acredito que me saí bem.	1	6%
Demorei para desenvolver meu primeiro programa, mas consegui.	0	0%
A linguagem é bastante difícil, preciso estudar melhor sua sintaxe.	0	0%

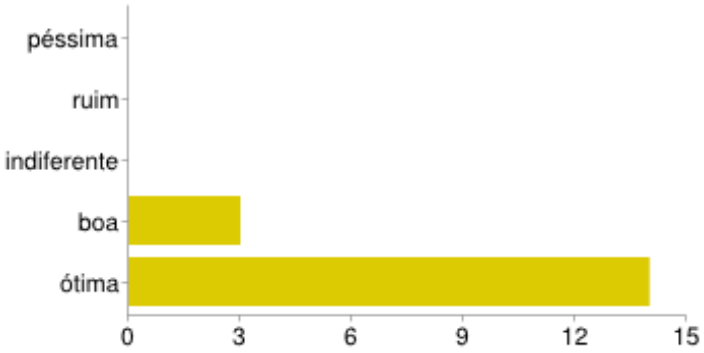
Qual a sua opinião em relação à utilização da ferramenta na disciplina de algoritmos?



A utilização da ferramenta seria interessante, facilitando o aprendizado dos alunos.	15	88%
A utilização da ferramenta seria indiferente.	1	6%
A utilização da ferramenta seria menos eficiente se comparada à formação que tive.	1	6%

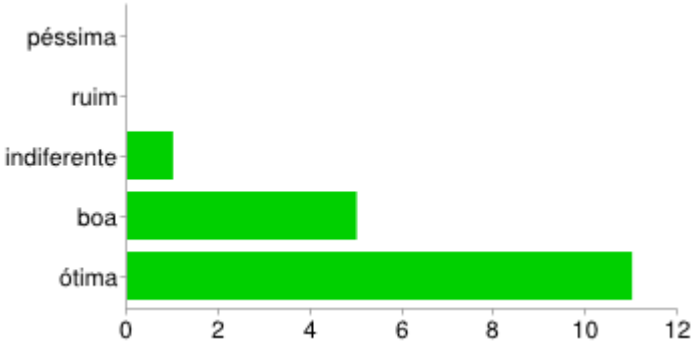
Como você avalia a contribuição de cada uma das funcionalidades da ferramenta para o ensino da disciplina de compiladores?

Executar a máquina passo a passo exibindo seu estado interno



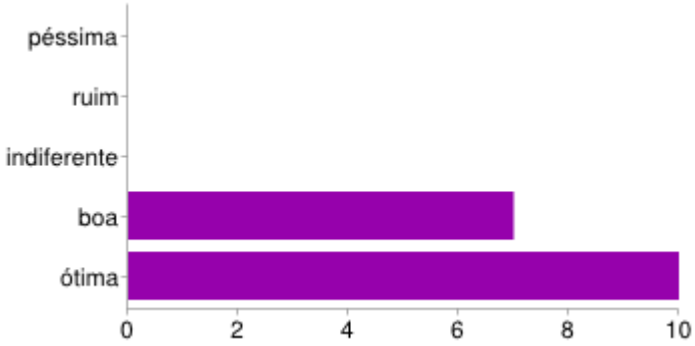
péssima	0	0%
ruim	0	0%
indiferente	0	0%
boa	3	18%
ótima	14	82%

Exibição da tabela de símbolos durante modos de depuração



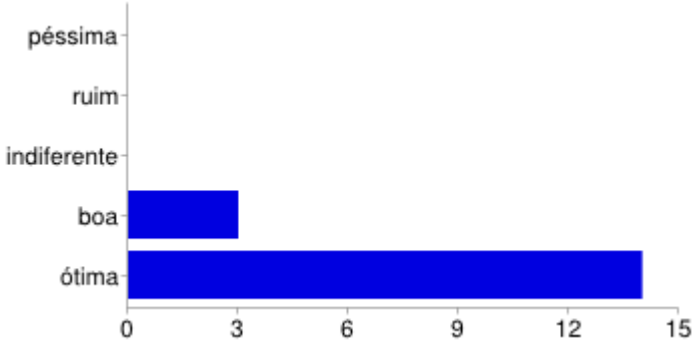
péssima	0	0%
ruim	0	0%
indiferente	1	6%
boa	5	29%
ótima	11	65%

Exibição da descrição dos microcódigos das instruções



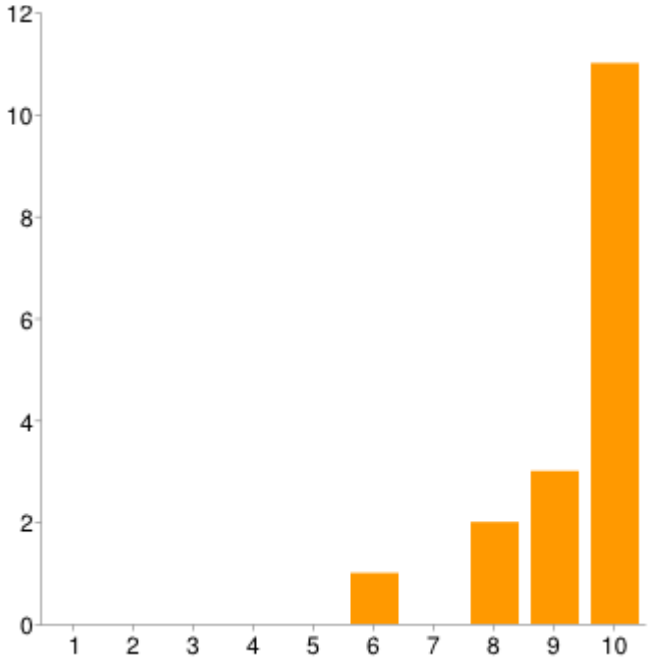
péssima	0	0%
ruim	0	0%
indiferente	0	0%
boa	7	41%
ótima	10	59%

Realce da linha de código fonte correspondente à instrução de máquina em execução



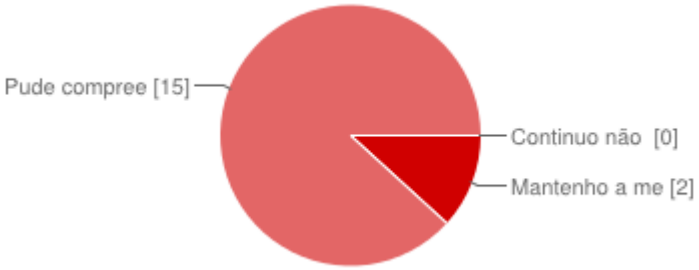
péssima	0	0%
ruim	0	0%
indiferente	0	0%
boa	3	18%
ótima	14	82%

Em uma escala, onde um é a pior nota e dez, a melhor. Como você classificaria a utilização da ferramenta na disciplina de compiladores?



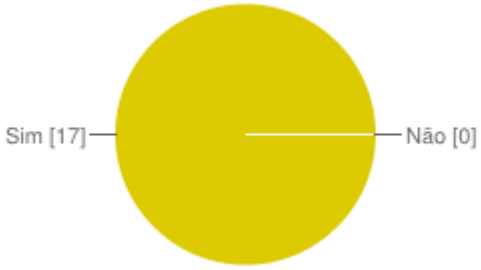
1	0	0%
2	0	0%
3	0	0%
4	0	0%
5	0	0%
6	1	6%
7	0	0%
8	2	12%
9	3	18%
10	11	65%

Depois de utilizar o simulador da Máquina Virtual Simples, como você avalia o seu entendimento sobre o funcionamento interno de um computador?



Mantenho a mesma visão que possuía.	2	12%
Pude compreender melhor a relação entre registradores, memória e instruções de máquina.	15	88%
Continuo não entendendo como um programa é executado por uma máquina.	0	0%

A partir da utilização da linguagem, da máquina e do compilador, propostos, você, graduando em ciência da computação, seria capaz de responder qual a relação entre a definição de uma linguagem de programação, a arquitetura da máquina utilizada e o compilador?



Sim	17	100%
Não	0	0%