# Implementação de um compilador: Trabalho 3

**Daniella Martins Vasconcellos**[1], **Matheus Soppa Geremias**[1]

[1]Departamento de Ciência da Computação – Universidade Estadual de Santa Catarina (UDESC)

daniellavasconc@gmail.com, suppersoppa@gmail.com

***Resumo.*** *Trabalho feito para a disciplina de Compiladores (COM0002), ministrada pelo Prof. Ricardo Ferreira Martins. O trabalho se divide nas seguintes sessões: desenvolvimento do cabeçalho e finalização do arquivo (aula "jvm"), geração de código intermediário, implementação dos comandos da linguagem, a execução do programa e o destaque do ambiente.*

## 1. Cabeçalho e finalização do arquivo

De forma simples, o cabeçalho é iniciado da forma abaixo, indicando os pacotes do Jasmin a serem carregados e utilizados e qual a main que ele está lendo; a única diferença de um cabeçalho para outro é o nome do arquivo de texto a ser lido e interpretado.

O método **escreveCodigo()** recebe uma string a ser escrita dentro da lista.

```
escreveCodigo(".source " + outfileName);
escreveCodigo(".class public test\n.super
   java/lang/Object\n");
escreveCodigo(".method public <init>()V");
escreveCodigo("aload_0");
escreveCodigo("invokenonvirtual java/lang/Object/<init>()V");
escreveCodigo("return");
escreveCodigo(".end method\n");
escreveCodigo(".method public static
   main([Ljava/lang/String;)V");
escreveCodigo(".limit locals 100\n.limit stack 100");
```

Para a finalização do arquivo, é escrito nas duas últimas linhas do código:

```
return
.end method
```

## 2. Geração de código intermediário

Aqui estão os comandos feitos para poderem percorrer o input. Os exemplos maiores de como cada estrutura é declarada será mostrada nos inputs na subsessão 3.1. Temos então:

- **inicio:** Fabricação do header.
- **marcador:** Escreve o código da label gerada em determinado ponto da leitura do código.
- **comando:** Responsável por chamar todos os outros comandos principais do código.

- **lista_comandos:** Responsável por chamar ou um único comando, ou um comando com um marcador, depois chamando a ele próprio de forma recursiva.
- **declaracao:** Para declarar variáveis. Pode ser de dois tipos:
  - **declaracao_normal:** Declaração de variável pura.
    Ex: int x;
  - **declaracao_valor:** Declaração de variável e associação de valor.
    Ex: int x = 10;
- **tipo:** Associação de tipos às variáveis. Três tipos aceitos: *int, double, boolean*.
- **atribuicao:** Responsável por fazer a atribuição de uma variável a seu tipo, terminado com um ponto e vírgula ao final.
- **expressao:** Chamada por outros comandos, é o comando que guarda informações se um token pode ser associado a um id, a uma expressão matemática ou a números. De forma recursiva, também pode chamar a ele próprio se necessário.
- **numeros:** Podem ser do tipo reais (ponto flutuante) ou inteiros.
- **expressao_matematica:** Executa operações matemáticas entre duas expressões.
- **printar:** Reconhecimento do token print(), que exibe informações na tela.
- **expressao_booleana:** Reconhece uma palavra booleana (true ou false) e pode fazer comparações entre expressões (do comando "expressao" acima) ou boleanas.
- **goto:** Aumenta o tamanho da lista do código e escreve "goto" no output, para indicar o caminho dentro de laços de repetição e estruturas condicionai.
- **if:** Pode chamar estrutura de um "se" sozinho ou "se... então".
  - **if_solo:** Pode ser um "se" declarado em uma única linha, ou em várias linhas.
    - **if_solo_linhas:** Estrutura condicional em mais de uma linha.
      Ex: *if (x == 10) {*
      *print x;*
      *}*
    - **if_solo_linha :** Estrutura condicional em linha única.
      Ex: *if (x == 10) print x;*
  - **if_else:** Estrutura condiconal "se" acompanhado por "senão".
- **for:** Estrutura de repetição do laço for. Baseado na linguagem C, abre-se um parênteses e deve-se primeiro declarar a variável que indica o início do laço, seguido por um ponto e vírgula. Depois, indica-se qual o limite do ciclo, seguido de outro ponto e vírgula. Por fim, indica-se de quanto em quanto deve ser executado o loop, que, diferentemente da linguagem base, também deve ser seguido de um ponto e vírgula antes do último colchete.
- **while:** Estrutura do laço de repetição "while". É executada de forma independente do comando do_while.
- **do_while:** Estrutura do laço de repetição "do". É executada de forma independente do comando while.

## 3. Implementação dos comandos da linguagem

A linguagem final implementada pelo compilador aceita os seguintes comandos:

- **Declaração de variáveis (inteiros ou ponto flutuante):** A linguagem precisa que o usuário escreva primeiro o tipo da variável e depois sua label para poder aceitá-la, utilizando apenas *int* e *float* para tal. É possível primeiro fazer uma declração vazia e depois atribuir um valor à variável escolhida.

- **Mostrar informações na tela:** Usa-se o comando *print()* para quando o usuário deseja retornar no terminal o valor de uma variável ou algum número.
- **Estruturas condicionais:** O compilador aceita estruturas de blocos que são executadas caso uma dada condição seja cumprida. Os blocos podem ser únicos (apenas um "if") ou encadeados ("if" ... "else").
- **Laços de repetição:** O compilador aceita dois tipos de laços de repetição: *"for ( ... ) { ... }"* e *"while { ... } do { ... }"*. Suas estruturas de implementação serão mostradas na seguinte subsessão.
- **Comentários:** Iniciados por "//", são ignorados pelo compilador.

### 3.1. Exemplos de código

Abaixo, estão alguns exemplos de inputs que o compilador pode aceitar.

#### 3.1.1. Input 1

```
int num;
num = 50;
if (num == 50) print(50);

int x = 10;
if (x >= 10) {
   print(10);
   print(333);
} else {
   print(-10);
   print(-333);
}

int y;
for ( x = 0 ; x < 2 ; x = x + 1;)
{
  for(y = x ; y < x + 5; y = y + 1;)
  {
    print(y);
  }
}

int i = 0;
while (i < 5) {
   print(i);
   i = i + 1;
}

do {
   print(i);
   i = i + 1;
} while (i < 0);
```

### 3.1.2. Input 2

```
double altura = 10.55;
double largura = 5.10;

//Isto eh um comentario

double area = 5.1*10.55;

if (50 > area) {
   double a = 3/4;
} else {
   double a = 5/4;
}
```

## 3.2. Dificuldades encontradas

Por mais que houvesse a tentativa de implementar o comando de *switch case*, não foi alcançada uma implementação perfeita. Dentro da geração do código intermediário, foram implementados os seguintes comandos não antes citados:

- **switch:** Identifica a estrutura condicional switch/case, chamando, entre parênteses depois do comando switch, uma variável.
- **default:** Definição do caso padrão, que é lido o token "default", seguido de dois pontos e a série de comandos.
- **case:** Definição dos outros casos, onde é lido o token "case", seguido do número do caso, dois pontos, e a lista de comandos associada.

Apesar de reconhecer as palavras e o formato da estrutura, primeiro o programa escreve o print no bytecote, em seguida entrando em um loop. Essa falha foi encontrada ao compilar o código abaixo.

```
int teste = 10;
switch (teste) {
   case 1: print(333);
   case 10: print(-666);
}
```

# 4. Execução do programa

Para que o trabalho seja executado, deve-se rodar os seguintes comandos no terminal:

*make*

Alternativamente, caso o usuário queira rodar cada comando que está dentro do makefile, ou se não possuir o makefile instalado, pode-se rodar os seguintes comandos:

*flex com.lex*
*bison -d com.y*

*g++ -std=c++11 com.tab.c lex.yy.c -o com -lm*
*./com*
*java -jar ./jasmin-1.1/jasmin.jar output.j*
*java test*

Além de compilar os códigos corretamente, provando o funcionamento da linguagem, ele também cria um arquivo chamado ***output.j***, que é o responsável por mostrar o funcionamento dos bytecodes e do que exatamente está sendo lido no código.

## 5. Destaque do ambiente

Esse projeto foi compilado utilizando um processador *Intel(R) Core(TM) i5-7200U CPU @ 2.50GHz*, com o sistema operacional *Linux Ubuntu 20.04 focal*, utilizando a versão do compilador *lex 1.5.003*, do *GNU Bison 3.5.1* e da biblioteca *Jasmin 1.1*. Lembrando que é necessária a instalação da biblioteca *makefile* para que o arquivo seja compilado e executado apenas com o comando "make".

## 6. Considerações finais

Agradecimentos ao Professor Ricardo Ferreira Martins pela disponibilização dos materiais utilizados como base para o estudo feito neste trabalho.