Centro Federal de Educação Tecnológica de Minas Gerais
Departamento de Computação – Engenharia de Computação – ***Compiladores***
Prof.ª Kecia Marques – 2019-1

# Implementação de um Compilador

O trabalho prático a ser realizado na disciplina de Compiladores é a construção de um compilador completo para uma linguagem de programação. O trabalho será realizado por etapas, conforme cronograma a seguir. Este documento especifica as características da linguagem e descreve as definições para a realização das demais etapas do trabalho.

## 1. Cronograma e Valor

O trabalho vale 30 pontos no total. Ele deverá ser entregue por etapas conforme o cronograma abaixo:

| ***Etapa*** | ***Data de entrega*** | ***Valor*** | ***Multa por atraso*** |
|---|---|---|---|
| 1 - Analisador Léxico e Tabela de símbolos | 30/04 | 8.0 | 1.0 |
| 2 - Analisador Sintático | 27/05 | 8.0 | 1.0 |
| 3 - Analisador Semântico | 17/06 | 8.0 | 1.0 |
| 4 – Gerador de código | 01/07 | 6.0 | - |

## 2. Regras

- O trabalho poderá ser realizado individualmente, em dupla ou em trio.
- Não é permitido o uso de ferramentas para geração do analisador léxico e do analisador sintático.
- A implementação deverá ser realizada em C/C++ ou Java. A linguagem utilizada na primeira etapa deverá ser a mesma para as etapas subsequentes. A mudança de linguagem utilizada ao longo do trabalho deverá ser negociada previamente com a professora.
- Realize as modificações necessárias na gramática para a implementação do analisador sintático.
- Não é necessário implementar recuperação de erro, ou seja, erros podem ser considerados fatais. Entretanto, as mensagens de erros correspondentes devem ser apresentadas, indicando a linha de ocorrência do erro.
- A organização do relatório será considerada para fins de avaliação.
- Trabalhos total ou parcialmente iguais receberão avaliação nula.
- Trabalhos total ou parcialmente iguais a projetos apresentados por outros alunos em semestres anteriores receberão avaliação nula (exceto se for o trabalho realizado exclusivamente pelo próprio aluno).

- A tolerância para entrega com atraso é de 1 semana, exceto no caso da Etapa 4 que não será recebida com atraso.
- Os trabalhos somente serão recebidos via Ava.
- A professora poderá realizar arguição com os alunos a respeito do trabalho elaborado. Nesse caso, a professora agendará um horário extraclasse para a realização da entrevista com o grupo.

### 3. Gramática da Linguagem

program ::= **app** identifier body
body ::= [ decl-list] **start** stmt-list **stop**
decl-list ::= decl {";" decl}
decl ::= type ident-list
ident-list ::= identifier {"," identifier}
type ::= **integer** | **real**

stmt-list ::= stmt {";" stmt}
stmt ::= assign-stmt | if-stmt | while-stmt | repeat-stmt
| read-stmt | write-stmt
assign-stmt ::= identifier ":=" simple_expr
if-stmt ::= **if** condition **then** stmt-list **end**
| **if** condition **then** stmt-list **else** stmt-list **end**
condition ::= expression

repeat-stmt ::= **repeat** stmt-list stmt-suffix
stmt-suffix ::= **until** condition
while-stmt ::= stmt-prefix stmt-list **end**
stmt-prefix ::= **while** condition **do**

read-stmt ::= **read** "(" identifier ")"
write-stmt ::= **write** "(" writable ")"
writable ::= simple-expr | literal

expression ::= simple-expr | simple-expr relop simple-expr
simple-expr ::= term | simple-expr addop term
term ::= factor-a | term mulop factor-a
fator-a ::= factor | "**!**" factor | "-" factor
factor ::= identifier | constant | "(" expression ")"
relop ::= "=" | ">" | ">=" | "<" | "<=" | "!="
addop ::= "+" | "-" | "**||**"
mulop ::= "*" | "/" | "**&&**"

constant ::= integer_const | float_const
integer_const ::= digit {digit}
float_const ::= digit {digit} "**.**" digit {digit}
literal ::= " **{**" {caractere} "**}**"
identifier ::= letter |"_" {letter | digit | "_"}
letter ::= [A-Za-z]
digit ::= [0-9]
caractere ::= *um dos 256 caracteres do conjunto ASCII, exceto "}"* e quebra de linha

## 4. Outras características da linguagem

- As palavras-chave da linguagem são reservadas.
- Toda variável deve ser declarada antes do seu uso. Uma variável pode ser declarada somente uma vez.
- Nenhuma variável pode ter o nome igual ao nome do programa.
- A entrada e a saída da linguagem estão limitadas ao teclado e à tela do computador.
- A linguagem possui somente comentários de uma linha que começa com "%"
- Os operadores são aplicáveis somente aos tipos numéricos.
- O resultado da divisão entre dois números inteiros é um número real.
- O comando de atribuição só é válido se os operandos possuírem o mesmo tipo.
- As operações de comparação resultam em valor lógico (verdadeiro ou falso)
- Nos testes (dos comandos condicionais e de repetição) a expressão a ser validada deve ser um valor lógico.
- A semântica dos demais comandos e expressões é a tradicional de linguagens como Pascal e C.
- A linguagem não é *case-sensitive*.
- O compilador da linguagem deverá gerar código a ser executado na máquina VM, que está disponível no Ava com sua documentação. A máquina VM é um arquivo executável para ambiente Windows.

## 5. O que entregar

Em cada etapa, deverão ser entregues via Ava:

- Código fonte do compilador.
- Código Java compilado ou C/C++ executável (para Windows e Linux).
- Relatório contendo:
  - Forma de uso do compilador
  - Descrição da abordagem utilizada na implementação, indicando as principais classes da aplicação e seus respectivos propósitos. Não deve ser incluída a listagem do código fonte no relatório.
  - Na etapa 2, as modificações realizadas na gramática

- Resultados dos testes especificados. Os resultados deverão apresentar o programa fonte analisado e a saída do Compilador: reportar sucesso ou reportar o erro e a linha em que ele ocorreu.
    - Na etapa 1, o compilador deverá exibir a sequência de tokens identificados e os símbolos (identificadores e palavras reservadas) instalados na Tabela de Símbolos. Nas etapas seguintes, isso **não** deverá ser exibido.
    - No caso de programa fonte com erro, o relatório deverá mostrar o código fonte analisado e o resultado indicando o erro encontrado. O código fonte deverá ser corrigido para aquele erro, o novo código e o resultado obtido após a correção deverão ser apresentados. Isso deverá ser feito para cada erro que o compilador encontrar no programa fonte.
- Na etapa 4, o código fonte analisado e seu respectivo código objeto gerado, bem como o resultado da execução do programa gerado na VM.

## 6. Testes

### Teste 1:

```
app teste1
  int a, b, c;
  int result
start
  read (a);
  read (c);
  b := 10.23 + a + c;
  result := (a + b) -;
  write(result);
  teste1 := 1
stop
```

### Teste 2:

```
app
  int a;
  int b;
  real f, _f
start
  read (a);
  read (f);
  f := a * a;
  b := b + a/2 * (3 + 5);
  write(f);
  write(b);
```

## Teste 3:

```
App pessoa
    int: cont;
    real altura, soma;
   start
      cont := 5;
      soma := 0;

      repeat

        write({Altura: });
        read (altura);
        soma := soma # altura;
        cont := cont - 1

      until(cont=0);
      media := soma / 5;

      write({Media: });
      write (media)

   stop
```

## Teste 4:

```
App teste4

    int i, j, k, @total, 1soma, teste4;
    float i, a
  start
      read (i);
      read (j);
      read (k);
      i := 4 * (5-3 * 50 / 10;
      j := i * 10;
      k := i* j / k;
      k := 4 + a $;
      write(i);
      write(j);
      write(k)
  stop
```

## Teste 5:

```
app Teste5

        int j, k;
        real a, j

   start
     read(j);
     read(k);
```

```
   if (k != 0)
      result := j/k
   else
      result := 0
   end;

   write(res)
stop
```

## Teste 6:

```
%Um programa para calcular a maior idade
int a, b, c, maior;
start
   read(a);
   read(b);
   read(c);

   maior = 0;


   if ( a>b && a>c  ) then
     maior := a;
   else
      if (b>c) then
          maior := b;
      else
          maior := c;
      end
   end

   write({Maior idade: );
   write(maior);
stop
```

## Teste 7:

Mostre mais dois testes que demonstrem o funcionamento de seu compilador.

**********