

## **Implementação de um Compilador**

O trabalho prático a ser realizado na disciplina de Compiladores é a construção de um compilador completo para um 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 cronograma abaixo:

<b><i>Etapas</i></b>	<b><i>Data de entrega</i></b>	<b><i>Valor</i></b>	<b><i>Multa por atraso</i></b>
1 - Analisador Léxico e Tabela de símbolos	28/abril	7.0	1.0
2 - Analisador Sintático	26/maio	8.0	1.0
3 - Analisador Semântico	23/junho	7.0	1.0
4 - Compilador completo, com geração de código	09/julho	8.0	-

### **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, a 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 Moodle.
- A professora poderá realizar arguição com os alunos a respeito do trabalho elaborado. Nesse caso, a professora agendará um horário extra-classe para a realização da entrevista com o grupo.

### 3. Gramática da Linguagem

program	::= [ <b>declare</b> decl-list ] <b>start</b> stmt-list <b>end</b>
decl-list	::= decl ";" { decl ";" }
decl	::= ident-list ":" type
ident-list	::= identifier { "," identifier }
type	::= <b>int</b>   <b>string</b>
stmt-list	::= stmt ";" { stmt ";" }
stmt	::= assign-stmt   if-stmt   do-stmt   read-stmt   write-stmt
assign-stmt	::= identifier "=" simple_expr
if-stmt	::= <b>if</b> condition <b>then</b> stmt-list <b>end</b>   <b>if</b> condition <b>then</b> stmt-list <b>else</b> stmt-list <b>end</b>
condition	::= expression
do-stmt	::= <b>do</b> stmt-list stmt-suffix
stmt-suffix	::= <b>while</b> condition
read-stmt	::= <b>read</b> "(" identifier ")"
write-stmt	::= <b>write</b> "(" writable ")"
writable	::= simple-expr
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	::= "+"   "-"   <b>or</b>
mulop	::= "*"   "/"   <b>and</b>
constant	::= integer_const   literal
integer_const	::= digit { digit }
literal	::= " " { caractere } " " "
identifier	::= letter { letter   digit }

letter	::= [A-Za-z]
digit	::= [0-9]
caractere	::= <i>um dos 256 caracteres do conjunto ASCII, exceto " e quebra de linha</i>

#### 4. Outras características da linguagem

- As palavras-chave da linguagem são reservadas.
- Toda variável deve ser declarada antes do seu uso.
- Nomes de identificadores podem ter no máximo 20 caracteres.
- O maior valor possível para uma constantes inteiras positiva é aquele aceito pela linguagem Java.
- A entrada e a saída da linguagem estão limitadas ao teclado e à tela do computador.
- A linguagem possui comentários de uma linha, que começa com com "//".
- A linguagem possui comentários de mais de uma linha que começa com "/\*" e termina com "\*/".
- Os operadores são aplicáveis somente aos tipos numéricos.
- O resultado da divisão entre dois números inteiros é um número inteiro.
- 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 Moodle com sua documentação. A máquina VM é um arquivo executável para ambiente Windows.

#### 5. O que entregar

Em cada etapa, deverão ser entregues via Moodle:

- 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:

```

program testel
  declare
    a, b, c: int;
    result: int
  start
    read (a);
    read (c);
    b = 10;
    result = (a * c) / (b + 5 - 345);
    write(result);
    testel = 1;
  end

```

### Teste 2:

```

declare
  a: int;
  b: int;
  nome: string;
start
  read (a);
  read (nome);
  b = a * a;
  b = b + a/2 * (3 + 5);
  write(nome);
  write(b);
end.

```

### Teste 3:

```
declare
    cont: int;
    media, altura, soma: int;
start
    cont = 5;
    soma = 0;

    do

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

    while(cont);

    write("Media: ");
    write (soma / qtd);

end
```

### Teste 4:

```
declare

    i, j, k, @total, lsoma: int;
start
    i = 4 * (5-3 * 50 / 10;
    j = i * 10;
    k = i* j / k;
    k = 4 + a $;
    write(i);
    write(j);
    write(k);
end
```

### Teste 5:

```
declare
    /*Este e um programa com if

        j, k: int;
        a, j: real;

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

```

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

    write(res);
end

```

## Teste 6:

```

declare
// Um programa para calcular a maior idade
a, b, c, maior: int;
umavariavelcomonomemuitogrande: string;
start
    read(a);
    read(b);
    read(c);

    maior = 12345678901234567890123456789;

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

    write("Maior idade: ");
    write(maior);
end

```

## Teste 7:

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

\*\*\*\*\*