Centro Federal de Educação Tecnológica de Minas Gerais
Departamento de Computação – Engenharia de Computação – ***Compiladores***
Prof.ª Kecia Marques – Segundo semestre de 2020

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 45 pontos no total. Ele deverá ser entregue por etapas conforme cronograma abaixo:

Etapa	***Data de entrega***	***Valor***
1 - Analisador Léxico e Tabela de símbolos	19/02/2021	15,0
2 - Analisador Sintático	19/03/2021	15,0
3 - Analisador Semântico e gerador de código	13/04/2021	15,0

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 3 que não será recebida com atraso.
- Os trabalhos somente serão recebidos via Moodle.

3. Gramática da Linguagem

program ::= **init** [decl-list] stmt-list **stop**
decl-list ::= decl ";" { decl ";"}
decl ::= ident-list **is** type
ident-list ::= identifier {"," identifier}
type ::= **integer** | **string | real**
stmt-list ::= stmt ";" { stmt ";" }
stmt ::= assign-stmt | if-stmt | do-stmt
| read-stmt | write-stmt
assign-stmt ::= identifier ":=" simple_expr
if-stmt ::= **if** "(" condition ")" **begin** stmt-list **end**
| **if** "(" condition ")" **begin** stmt-list **end else**
begin stmt-list **end**
condition ::= expression

do-stmt ::= **do** stmt-list do-suffix
do-suffix ::= **while** "(" condition ")"

read-stmt ::= **read** "(" identifier ")"
write-stmt ::= **write** "(" 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
factor-a ::= factor | **not** factor | "-" factor
factor ::= identifier | constant | "(" expression ")"
relop ::= "=" | ">" | ">=" | "<" | "<=" | "<>"
addop ::= "+" | "-" | **or**
mulop ::= "*" | "/" | **and**

Padrão de formação dos tokens

constant → integer_const | literal
integer_const → nonzero {digit} | “0”
real_const → interger_const "." $digit^{+}$

literal → " “ " caractere* " ” "
identifier → letter {letter | digit | " _ " }
letter → [A-Za-z]
digit → [0-9]
nonzero → [1-9]
caractere → *um dos 256 caracteres do conjunto ASCII, exceto* as aspas 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.
- A entrada e a saída da linguagem estão limitadas ao teclado e à tela do computador.
- A linguagem possui comentários que começam com “%” e termina com “%”.
- Os operadores aritméticos 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 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.
- Se desenvolvido em Java, entregar o JAR também.
- 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.

- Em cada etapa, deverão ser considerados os códigos fontes sem erros da última etapa realizada até então. Por exemplo, na etapa 2, Análise Sintática, os códigos fontes a serem considerados são aqueles sem os possíveis erros léxicos reportados na etapa 1.
- Na etapa 3, 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:

```
init
  a, b, $c, is integer;
  result is real;

  write("Digite o valor de a:"
  read (a);
  write("Digite o valor de c:"
  read (c);
  b := 10
  result := (a * c)/(b + 5 - 345);
  write("O resultado e:");
  write(result);

stop
```

Teste 2:

```
  a, _valor, b_1, b_2 : integer;

init
  write("Entre com o valor de a:");
  read (a);
  b_1 := a * a;
  write("O valor de b1 e:");
  write (b_1);
  b_2 = b + a/2 * (a + 5);
  write("O valor de b1 e:");
  Write (b1);
```

```
stop
```

Teste 3:

```
% Programa de Teste
Calculo de idade%

INIT
    cont is int;
    media, idade, soma is integer;
   begin
      cont := 5;
      soma := 0;

      do

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

      while(cont > 0)
      media := soma / qts
      write("Media: ");
      write (media);

 STOP
```

Teste 4:

```
init
% Outro programa de teste
    i, j, k, @total is integer;
    nome is string

      write("Digite o seu nome: ");
      read(nome);
      write("Digite um valor inteiro: );
      read (I);
      k := i * (5-i * 50 / 10;
      j := i * 10;
      k := i* j / k;
      k := 4 + a $;
      write(nome);
      write(", os números gerados sao: ");
      write(i);
      write(j);
      write(k);
```

Teste 5:

```
init

    i, j, k, @total is integer;
    nome is string
```

```
write("Digite o seu nome: ");
read(nome);
write("Digite um valor inteiro: );
read (I);
k := i * (5-i * 50 / 10;
j := i * 10;
k := i* j / k;
k := 4 + a $;
write(nome);
write(", os números gerados sao: ");
write(i);
write(j);
write(k);
```

Teste 6:

```
init
   a, b, c, maior is integer;

      write("Digite uma idade: ");
      read(a);
      write("Digite outra idade: ");
      read(b);
      write("Digite mais uma idade: ");
      read(c;

      maior := 0;


      if ( a>b and a>c  )
        maior := a;

      else
         if (b>c)
             maior := b;

         else
             maior := c;



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

Teste 7:

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

****