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

Trabalho Prático

Este trabalho prático tem por objetivo a construção de um compilador para uma linguagem de programação especificada.

1. Valor

O trabalho vale 40 pontos no total. Ele deverá ser entregue por etapas conforme cronograma abaixo:

Etapa	Data de entrega	Valor
1 - Analisador Léxico e Tabela de símbolos	25/09	10
2 - Analisador Sintático	23/10	15
3 - Analisador Semântico e Gerador de Código	04/12	15

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.
- 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.
- A tolerância para entrega com atraso é de 1 semana, exceto no caso da Etapa 3 que não será recebida com atraso.

3. Gramática

program ::= **program** identifier **is** body
body ::= [**declare** decl-list] **init** stmt-list **end.**
decl-list ::= decl {";" decl}
decl ::= ident-list ":" type
ident-list ::= identifier {"," identifier}

type ::= **int** | **float** | **char**

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 ::= **in** "<<" identifier
write-stmt ::= **out** ">>" 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 | char_const
integer_const ::= digit {digit}
float_const ::= digit{digit} “**.**”digit{digit}
char_const ::= " ‘ " carac " ’ "
literal ::= " **“**" {caractere} "**”**"
identifier ::= letter {letter | digit | “_”}
letter ::= [A-Za-z]
digit ::= [0-9]
carac ::= *um dos caracteres ASCII*
caractere ::= *um dos caracteres ASCII, exceto* **“”** e quebra de linha

4. Outras características da linguagem

- As palavras-chave são reservadas.
- Toda variável deve ser declarada antes do seu uso.
- Entrada e saída de dados estão limitadas ao teclado e ao monitor.
- A linguagem possui comentários de mais de uma linha. Um comentário começa com "/*" e deve terminar com "*/"
- É possível atribuir um dado do tipo inteiro a uma variável do tipo float, mas o inverso não é permitido. Nos demais casos, os tipos são incompatíveis.
- A linguagem é *case-sensitive*.
- O compilador da linguagem deverá gerar código a ser executado na máquina VM, que está disponível no Moodle com sua documentação.

5. O que entregar?

Em cada etapa, deverão ser entregues via Moodle:

- Código fonte do compilador.
- No caso de Java, o código compilado (JAR).
- 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 os erros e as linhas em que eles ocorreram.
    - 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 3, deverá ser mostrado todos os resultados da análise semântica, o código fonte analisado e seu respectivo código objeto gerado, bem como o resultado da execução do programa gerado na VM.

6. Testes

Em cada etapa, os programas a seguir deverão ser analisados pelo Compilador. Os erros identificados em uma etapa devem ser corrigidos para o teste da etapa seguinte. Por exemplo, os erros léxicos, identificados na etapa 1, devem ser corrigidos no programa antes de ele ser submetido ao compilador obtido na etapa 2.

Teste 1:

```
programa teste1 is

   a, b: int;
   result : int;
   a,x : float
begin
  a = 12a;
  x = 12.;
  in << a;
  in << b;
  in << c
  result = (a*b + 1) / (c+2);
  out "Resultado: ";
  out >> result;
end.
```

Teste 2:

```
program teste2 is
 declare
     a, b, c:int;
     d, _var: float;
begin
  teste2 = 1;
  In << a;
  b = a * a;
  c = b + a/2 * (35/b);
  out c;
  val := 34.2
  c := val + 2.2;
  out >> val
```

Teste 3:

```
program is
  declare
       a, aux: int;
       b: float;
  begin
     b = 0;
     in <<a;
     in<<b;
     if (a>b) then
       aux = b;
       b = a;
       a = aux
     end;
     out >> a;
     out >> b
  end
```

Teste 4:

```
programa teste4

declare
    pontuacao, pontuacaoMaxina, disponibilidade: inteiro;
    pontuacaoMinima: char
begin
  pontuacaoMinima = 50;
  pontuacaoMaxima = 100;
  out << "Pontuacao Candidato:";
  in << pontuacao;
  out >> "Disponibilidade Candidato: ");
  in << disponibilidade;


  while (pontuacao>0 && (pontuação<=pontuacaoMaxima) do
     if ((pontuação > pontuacaoMinima) && (disponibilidade==1)) then
        out >> "Candidato aprovado")
     else
        out >>"Candidato reprovado")
     end

     out>>"Pontuacao Candidato: ";
     i << pontuacao;
     out>>"Disponibilidade Candidato: ";
     in << disponibilidade;

  end
end.
```

Teste 5:

```
program teste5
   a, b, c, maior, outro: int;
   begin
   repeat
      out(“A”);
      in(a);
      out(“B”);
      in(b);
      out(“C”);
      in(c);



      if ( (a>b) and (a>c)  ) end
        maior = a

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

         else
             maior = c
         end
      end;
      out(“Maior valor:””);
      out (maior);
      out (“Outro?”);
      in(outro);
   until (outro == 0)
   end.
```

Teste 6:

Mostre um teste contendo o comando repeat-until.

****