Centro Federal de Educação Tecnológica de Minas Gerais
Departamento de Computação – Engenharia de Computação – ***Compiladores***
Prof.ª Kecia Marques – Segundo semestre de 2025

Trabalho Prático

Este trabalho prático tem por objetivo a construção de um compilador.

1. Valor

O trabalho vale 30,0 pontos no total. Ele deverá ser entregue por etapas.

Etapa	*Valor*	*Entrega*	*Tolerância*
1 - Analisador Léxico	10,0	25/05/2025	01/06/2025
2 - Analisador Sintático	10,0	22/06/2025	29/06/2025
3 - Analisador Semântico e Tabela de Símbolos	10,0	25/07/2025	-

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.
- A tolerância para entrega com atraso é de uma semana.
- 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 extraclasse para a realização da entrevista com o grupo.

3. Gramática

program ::= **program** [decl-list] **begin** stmt-list **end**
decl-list ::= decl {decl}
decl ::= type “:” ident-list “;”
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** [decl-list] stmt-list **end**
| **if** condition **then** [decl-list] stmt-list **else**
declaration stmt- list **end**
condition ::= expression

repeat-stmt ::= **repeat** [decl-list] stmt-list stmt-suffix
stmt-suffix ::= **until** condition
while-stmt ::= stmt-prefix [decl-list] 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

Definição regular de tokens:

integer_const → digit^{+}
float_const → 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, que começa com "{" e deve terminar com "}".
- A linguagem possui comentários de uma linha, que começa com "%" e deve terminar com a primeira ocorrência de quebra de linha.
- O comando de atribuição somente é permitido quando o tipo da expressão for igual ao da variável.
- Nas operações aritméticas, quando os operandos forem numéricos e um deles for real, o resultado é um real. Quando ambos os operandos forem inteiros, o resultado é um inteiro. Quando um dos operandos for caractere e outro for inteiro, o resultado é um inteiro. Nesse caso, o valor caractere é convertido para o seu respectivo código ASCII. Os tipos caractere e real são incompatíveis.
- A linguagem não é *case-sensitive*.

Em cada etapa, deverão ser entregues via Moodle:

- Código fonte do compilador.
- Código Java compilado (caso tenha sido implementado em Java).
- 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. 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.

5. 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.

Teste 1:

```
program
  int: a,b,c;
  float: result;
  char: ch;

begin
  out("Digite o valor de a:");
  in (a);
  out("Digite o valor de c:");
  read (ch);
  b = 10;
  result = (a * ch)/(b  5 - 345.27);
  out("O resultado e: ");
  out(result);
  result = result + ch;
end
```

Teste 2:

```
program

     float: raio, area$ = 0.0;

begin
repeat
   in(raio);
   char: resposta;
   if (raio > 0.0) then
     area =  3. * raio * raio;
     out (area);
   end;
   out ("Deseja continuar?);
   in (resp);

until (resp == 'N' || resp == 'n');

end
```

Teste 3:

```
program

  int: a, b, aux;

  begin
     in (a);
     in(b);
     if (a>b) then
      int aux;
      aux = b;
      b = a;
      a = aux
     end;
     out(a;
     out(b)
  end
```

Teste 4:

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

      %Verifica o maior

      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 5:

```
programa

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

  { Comentario
  grande

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

     out("Pontuacao Candidato: ");
     in(pontuacao);
     out("Disponibilidade
Candidato: ");
     in(disponibilidade);

  end
  out (cont);
end
```

Teste 6:

Mostre mais um teste que demonstre o funcionamento de seu analisador léxico.

****