Centro Federal de Educação Tecnológica de Minas Gerais
Departamento de Computação – Engenharia de Computação – ***Compiladores***
Prof.ª Kecia Marques – 2023/1

Implementação de um Compilador

O trabalho prático a ser realizado na disciplina de Compiladores é a construção de um compilador 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 40 pontos no total. Ele deverá ser entregue por etapas conforme cronograma abaixo:

Etapa	***Valor***	***Entrega***	***Limite***
1 - Analisador Léxico e Tabela de símbolos	10	09/04	16/04
2 - Analisador Sintático	10	21/05	28/05
3 - Analisador Semântico	10	18/06	25/06
4 - Gerador de Código	10*	07/07	-

*O valor do trabalho é 35,0, porém, há 5,0 pontos-extras no gerador de código.

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 de forma clara, 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 tiver sido 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.
- 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 ::= **program** identifier **begin** [decl-list] stmt-list **end** "."
decl-list ::= decl ";" { decl ";"}
decl ::= ident-list **is** 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 ::= **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 | char_const

Padrões dos tokens

digit ::= [0-9]
carac ::= *um dos caracteres ASCII*
caractere ::= *um dos caracteres ASCII, exceto* quebra de linha
integer_const ::= digit^{+}

float_const ::= digit^{+} “.”digit^{+}
char_const ::= " ‘ " carac " ’ "
literal ::= "**{**" caractere* "**}**"
identifier ::= letter (letter | digit | " _")*
letter ::= [A-Za-z]

char_const e literal tem que ser tipos de tokens ou podem ser tratados apenas no Analisador Sintático???

Quando eu tenho
int x = 10
Eu crio o token x com valor 'x', quando durante a compilacao que o valor 10 vai ser atribuido para x e colocado como valor de x na TS???

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.
- 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.
- O resultado de uma divisão é sempre um float.
- A linguagem é *case-sensitive*.
- O compilador da linguagem deverá gerar código a ser executado na máquina VM ou para Jasmin (http://jasmin.sourceforge.net/). VM 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.

- Na geração de código, deverão ser entregues o código fonte analisado e seu respectivo código objeto gerado, bem como o resultado da execução do programa gerado na VM ou em Java.

- 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.

6. Testes

Teste 1:

```
programa teste1

   a, b is int;
   result is int;
   a,x is float;

begin

  a = 12a;
  x = 12.;
  read (a);
  read (b);
  read (c)
  result = (a*b + 1) / (c+2);
  write {Resultado: };
  write (result);

end.
```

Teste 2:

```
program teste2

     a, b, c:int;
     d, _var: float;

  teste2 = 1;
  Read (a);
  b = a * a;
  c = b + a/2 * (35/b);
  write c;
  val := 34.2
  c = val + 2.2 + a;
  write (val)
end.
```

Teste 3:

```
program
  a, aux is int;
  b is float

  begin
     b = 0;
     in (a);
     in(b);
     if (a>b) then  //troca variaveis
       aux = b;
       b = a;
       a = aux
     end;
     write(a;
     write(b)
```

Teste 4:

```
programa teste4

/* Teste4 do meu compilador

    pontuacao, pontuacaoMaxina, disponibilidade is inteiro;
    pontuacaoMinima is char;

begin
  pontuacaoMinima = 50;
  pontuacaoMaxima = 100;
  write({Pontuacao do candidato: });
  read(pontuacao);
  write({Disponibilidade do candidato: });
  read(disponibilidade);


  while (pontuacao>0 & (pontuacao<=pontuacaoMaxima) do
     if ((pontuação > pontuacaoMinima) && (disponibilidade==1)) then
        write({Candidato aprovado.})
     else
        write({Candidato reprovado.})
     end

     write({Pontuacao do candidato: });
     read(pontuacao);
     write({Disponibilidade do candidato: });
     read(disponibilidade);

  end
end
```

Teste 5:

```
/* Teste do meu compilador */

program teste5
   a, b, c, maior is int;
   outro is char;

begin
   repeat
      write({A});
      read(a);
      write({B});
      read(b);
      write({C});
      read(c);



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

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

         else
             maior = c
         end
      end;
      write({Maior valor:}});
      write (maior);
      write ({Outro? (S/N)});
      read(outro);
   until (outro == 'N' || outro == 'n)
end
```

Teste 6:

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