Centro Federal de Educação Tecnológica de Minas Gerais
Departamento de Computação – Engenharia de Computação – ***Compiladores***
Prof.ª Kecia Marques – Primeiro semestre de 2016

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

Etapa	***Data de entrega***	***Valor***	***Multa por atraso***
1 - Analisador Léxico e Tabela de símbolos	25/abril	7.0	1.0
2 - Analisador Sintático	16/maio	8.0	1.0
3 - Analisador Semântico	13/junho	8.0	1.0
4 – Compilador completo, com geração de código	01/julho	7.0	-

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 ::= [**var** decl-list] **begin** stmt-list **end**
decl-list ::= decl ";" { decl ";"}
decl ::= ident-list **is** type
ident-list ::= identifier {"," identifier}
type ::= **int** | **string**
stmt-list ::= stmt ";" { stmt ";"}
stmt ::= assign-stmt | if-stmt | do-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

do-stmt ::= **do** stmt-list stmt-suffix
stmt-suffix ::= **while** condition

read-stmt ::= **in** "(" identifier ")"
write-stmt ::= **out** "(" 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 | **not** factor | "-" factor
factor ::= identifier | constant | "(" expression ")"
relop ::= "=" | ">" | ">=" | "<" | "<=" | "<>"
addop ::= "+" | "-" | **or**
mulop ::= "*" | "/" | **and**
constant ::= integer_const | literal
integer_const ::= nozero {digit} | “0”
literal ::= " **{**" {caractere} "**}**"
identifier ::= (letter) {letter | digit } |

	“_” (letter | digit) {letter | digit }
letter	::= [A-Za-z]
digit	::= [0-9]
nozero	::= [1-9]
caractere	::= *um dos 256 caracteres do conjunto ASCII, exceto* “{”, “}” 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.
- Nomes de identificadores podem ter no máximo 15 caracteres.
- 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 “//”.
- A linguagem possui comentários de mais de uma linha que começa com “%” e termina com “%”.
- Os operadores aritméticos 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 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:

```
var
  a, b, c is int;
  result is int
begin
  in (a);
  in (c);
  b := 10;
  result := (a * c)/(b + 5 - 345);
  out(result);
end
```

Teste 2:

```
  a, _ is int;
  b is int;
  nome is string;
begin
  in (a);
  in (nome);
  b := a * a;
  b := b + a/2 * (3 + 5);
  out (nome);
  out(b);
end.
```

Teste 3:

```
var
    _cont is int;
    media, altura, soma_ is int;
   begin
      _cont := 5;
      soma = 0;

      do

        write({Altura: });
        in (altura);
        soma := soma  altura;
        _cont := _cont - 1;

      while(_cont);

      out({Media: });
      out (soma / qtd);

   end
```

Teste 4:

```
var

    i, j, k, @total, 1soma is int;
  begin
      i := 4 * (5-3 * 50 / 10;
      j := i * 10;
      k := i* j / k;
      k := 4 + a $;
      out(i);
      out(j);
      out(k);
  end
```

Teste 5:

```
var

        j, k is int;
        a, j real;

   begin
     read(j);
     read(K);

     if (k <> 0)
        result := j/k
     else
        result := 0
```

```
    end;

    out(result);
  end
```

Teste 6:

```
var
   a, b, c, maior is int;
   nomecompletodoalunodeposgraduacao is string;
   start
      read(a);
      read(b);
      read(c;

      maior := 0;


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

      Out({Maior idade: });
      out(maior);
   end
```

Teste 7:

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

****