

Trabalho Prático

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

1. Valor

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

<i>Etapa</i>	<i>Valor</i>	<i>Multa por atraso</i>	<i>Entrega</i>
1 - Analisador Léxico e Tabela de símbolos	8.0	1,0	04/10
2 - Analisador Sintático	8.0	1,0	01/11
3 - Analisador Semântico	8.0	1,0	22/11
4 - Gerador de Código	8,0 *	-	07/12

* 6,0 pontos + 2,0 extras

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, as 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 1 semana, exceto no caso da Etapa 4 que não será recebida com atraso.
- Os trabalhos somente serão recebidos via AVA.
- 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

program	::= start [decl-list] stmt-list exit
decl-list	::= decl {decl}
decl	::= type ident-list ";"
ident-list	::= identifier {" , " identifier}
type	::= int float string
stmt-list	::= stmt {stmt}
stmt	::= assign-stmt ";" if-stmt while-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
while-stmt	::= do stmt-list stmt-suffix
stmt-suffix	::= while condition end
read-stmt	::= scan "(" identifier ")"
write-stmt	::= print "(" 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 not factor "-" factor
factor	::= identifier constant "(" expression ")"
relop	::= "==" ">" ">=" "<" "<=" "<>"
addop	::= "+" "-" or
mulop	::= "*" "/" and
constant	::= integer_const float_const literal
integer_const	::= digit {digit}
float_const	::= digit{digit} "." digit{digit}
literal	::= " " " {caractere} " " "
identifier	::= letter {letter digit }
letter	::= [A-za-z]
digit	::= [0-9]
caractere	::= <i>um dos caracteres ASCII, exceto " (aspas) e quebra de linha</i>

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 “*/”.
- A linguagem possui comentários de uma linha. Um comentário começa com “//” e deve terminar com a primeira ocorrência de quebra de linha.
- Somente operandos de mesmo tipo são compatíveis.
- No caso do operador “+”, quando ambos operandos forem inteiros, ocorre uma soma entre os valores, sendo o resultado também do tipo inteiro. Quando ambos operandos forem do tipo *string*, o resultado é uma nova *string* que corresponde à concatenação dos operandos.
- 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 AVA:

- 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

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:

```
start
  int a, b;
  int result;
  float a,x,total;

  a = 2;
  x = .1;
  scan (b);
  scan (y)
  result = (a*b ++ 1) / 2;
  print "Resultado: ";
  print (result);
  print ("Total: ");
  total = y / x;
  print ("Total: ");
  print (total);
exit
```

Teste 2:

```
start

  int: a, c;
  float d, _e;

  a = 0; d = 3.5
  c = d / 1.2;

  Scan (a);
  Scan (c);
  b = a * a;
  c = b + a * (1 + a*c);
  print ("Resultado: ");
  print c;
  d = 34.2
  e = val + 2.2;
  print ("E: ");
  print (e);
  a = b + c + d)/2;
```

Teste 3:

```
int pontuacao, pontuacaoMaxima, disponibilidade;
string pontuacaoMinima;

disponibilidade = "Sim";
pontuacaoMinima = 50;
pontuacaoMaxima = 100;

/* Entrada de dados
  Verifica aprovação de candidatos
do
  print("Pontuacao Candidato: ");
  scan(pontuacao);
  print("Disponibilidade Candidato: ");
  scan(disponibilidade);

  if ((pontuação > pontuacaoMinima) and (disponibilidade=="Sim") then
    out("Candidato aprovado");
  else
    out("Candidato reprovado")
  end
  while (pontuação >= 0)end
exit
```

Teste 4:

```
start

    Int a, aux$, b;
    string nome, sobrenome, msg;

    print(Nome: );
    scan (nome);
    print("Sobrenome: ");
    scan (sobrenome);
    msg = "Ola, " + nome + " " +
sobrenome + "!";
    msg = msg + 1;
    print (msg);

    scan (a);
    scan(b);
    if (a>b) then
        aux = b;
        b = a;
        a = aux;
    end;
    print ("Apos a troca: ");
    out(a);
    out(b)
exit
```

Teste 5:

```
start
    int a, b, c, maior, outro;

    do
        print("A");
        scan(a);
        print("B");
        scan(b);
        print("C");
        scan(c);
        //Realizacao do teste

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

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

            else
                maior = c;
            end
        end
        print("Maior valor:");
        print (maior);
        print ("Outro? ");
        scan(outro);
        while (outro >= 0);
    exit
```

Teste 6:

Mostre mais um teste que demonstre o funcionamento de seu compilador.
