

Implementação de um Compilador

O trabalho prático a ser realizado na disciplina de Compiladores é a construção do *front-end* 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:

<i>Etapas</i>	<i>Valor</i>	<i>Entrega</i>	<i>Limite</i>
1 - Analisador Léxico e Tabela de símbolos	10,0	23/05	30/05
2 - Analisador Sintático	10,0	13/06	20/06
3 - Analisador Semântico	10,0	08/07	

2. Regras

- O trabalho poderá ser realizado individualmente ou em dupla.
- 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.
- 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 3 que não será recebida com atraso.
- Os trabalhos serão recebidos somente via Moodle.

3. Gramática da Linguagem

program	::= routine body
body	::= [decl-list] begin stmt-list end
decl-list	::= declare 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 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 not factor "-" factor
factor	::= identifier constant "(" expression ")"
relop	::= "=" ">" ">=" "<" "<=" "<>"
addop	::= "+" "-" or
mulop	::= "*" "/" and
constant	::= integer_const float_const char_const

Padrões dos tokens

integer_const	::= digit ⁺
float_const	::= digit ⁺ "." digit ⁺
char_const	::= " ' " char " ' "
literal	::= " " caractere* " "
identifier	::= letter (letter digit) [*]
letter	::= [A-Za-z]
digit	::= [0-9]
charac	::= <i>um dos caracteres ASCII</i>

caractere ::= *um dos caracteres ASCII, exceto quebra de linha e aspas*

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.
- O resultado de uma divisão é sempre um float.
- A linguagem é *case-sensitive*.

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.
 - 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:

```
routine

declare
    int a, b;
    int resul;
    float a,x;

begin

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

end
```

Teste 2:

```
routine

    int a, b, c;
    float d, _var
begin
    read (a);
    b := a * a;
    c := b + a/2 * (35/b); %aplica formula
    write c;
    val := 34.2
    c = val + 2. + a;
    write (val)
end
```

Teste 3:

```
routine
declare
    int a, aux;
    float b;

    begin
        B := 0;
```

```

read (a);
read(b);
if (a<>) b then
begin

    if (a>b) then
        aux := b;
        b := a;
        a := aux;
    end;
    write(a;
    write(b)
end
else
    write("Numeros iguais.");
end

```

Teste 4:

```

routine

declare
    int pontuacao, pontuacaoMaxima, disponibilidade;
    char pontuacaoMinima;

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

%
Processamento
%
    while (pontuacao>0 and (pontuacao<=pontuacaoMaxima) do
        if ((pontuação > pontuacaoMinima) and (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:

```
declare
    integer a, b, c, maior;
    char outro;

begin
    repeat
        write("A: ");
        read(a);
        write("B: ");
        read(b);
        write("C: ");
        read(c);

        if ( (a>b) and (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.