Projeto de Compiladores

1. Introdução

Este documento descreve a linguagem *Algox*, uma linguagem de algoritmo hipotética, que será utilizada como exemplo na demonstração dos conceitos envolvidos na introdução a compiladores. A linguagem *Algox* é simples e permite descrever algoritmos básicos envolvendo operações aritméticas e comandos de entrada e saída em um console.

2. Estrutura geral do programa

O programa nesta linguagem tem um único corpo da forma:

PROGRAMA <nome_do_programa>
INICIO
<corpo_do_programa>
FIM

O corpo do programa é dividido em duas áreas. A primeira área consiste de declarações de variáveis, e a segunda consiste do algoritmo propriamente dito, conforme a seguir. Trechos entre os caracteres "{" e "}" e o fim de linha são consideradas comentários.

{ Área de declarações }
DECLARACOES
{ Algoritmo }
ALGORITMO

Uma declaração de variável segue o formato TIPO NOME onde NOME é uma sequência qualquer de letras e números, sendo que o primeiro caractere deve ser uma letra, e TIPO é "INTEIRO" ou "REAL" ou "CARACTER" ou "CADEIA" ou "LISTA_INT" ou "LISTA_REAL".

Por exemplo:

INTEIRO var1	**(número inteiro)**
REAL var2	**(número real)**
CARACTER A, B	**(um carácter)**
CADEIA C[30], D[10]	**(sequencia de símbolos)**
LISTA_INT vetor[10]	**(arranjo de inteiros)**
LISTA_REAL VetReal[50]	**(arranjo de reais)**

O algoritmo consiste de uma sequência de comandos, descritos mais adiante. Espaços em branco e fim de linha não têm significado.

3. Expressões

Os comandos da linguagem *T*- podem fazer uso de expressões. Existem quatro tipos de expressões: expressões aritméticas, expressões relacionais e, expressões lógicas.

Uma expressão aritmética pode ser: uma constante inteira ou real, uma variável, ou uma operação aritmética (soma, subtração, multiplicação, divisão ou resto da divisão, somente para inteiros) envolvendo duas expressões.

As operações aritméticas seguem as regras convencionais de precedência e associatividade, isto é, todos operadores são associativos à esquerda, e os operadores "*" e "/" tem precedência sobre "+" e "-". Parêntesis podem ser utilizados para forçar a precedência.

Exemplos de expressões aritméticas são:

1000	**(constante inteira)**
3,14	**(constante real)**
var1	**(variável)**
var1 + 5	**(operação aritmética)**
2+3*5	**(operações aritméticas compostas)**
(2 + 3) * 5	**(operações aritméticas compostas)**

Uma *expressão relacional* envolve apenas dois operadores relacionais:

menor ou igual (<=) anotado como ".M." e,
igual (=) anotado como ".I.",

Esses operadores tem a mesma precedência e são associativos à esquerda. Os operadores relacionais só podem ser utilizados entre duas expressões aritméticas.

Operadores booleanos “ou” e “e” (anotados por **.O.** e **.E.**) podem ser utilizados para criação de expressões relacionais compostas, e o *operador booleano de negação* (anotado por **.N.**), para criação de expressão relacional simples.

Os operadores relacionais têm precedência sobre os operadores booleanos. Parêntesis podem ser utilizados para forçar a precedência entre dois operadores booleanos.

Exemplos de expressões relacionais são:

```
var1 .M. Var2                          ( comparação envolvendo duas variáveis)
var1 * var2 .M. var3                   (comparação entre operação aritmética e uma variável)
var1 .M. var2 .OU. var1 .M. var3       (expressão relacional composta)
.N. (var1 .I. Var2)                    (expressão relacional simples)
```

4. Comandos

Os comandos na linguagem *T-* possibilitam ações de atribuição, entrada, saída, seleção e repetição criadas pelo programador.

O *comando de atribuição*, caracterizado pela notação “**:=**”, armazena um valor em uma variável.

Segue o formato **VARIÁVEL := EXPRESSÃO**. Por exemplo:

```
var1 := 1000
var3 := var1 + var2
```

O *comando de entrada*, caracterizado pela notação “**LEIA**”, faz a leitura do usuário e armazena o valor lido em uma variável. Segue o formato LEIA VARIÁVEL ou LEIA VARIÁVEL1, VARIÁVEL2, ... Por exemplo:

```
LEIA var1
LEIA var3
LEIA var1, var2, var3
```

O *comando de saída*, caracterizado pela notação “**ESCREVA**”, imprime o valor de uma variável ou uma constante do tipo cadeia de caracteres no console. Segue o formato ESCREVA VARIÁVEL ou ESCREVA CADEIA ou ESCREVA CADEIA, VARIÁVEL ou ESCREVA VARIÁVEL, CADEIA.

Uma cadeia é uma sequência de caracteres delimitados por aspas simples, que não pode extrapolar uma linha. Por exemplo:

```
ESCREVA var1
ESCREVA var3
ESCREVA 'Alo mundo'
ESCREVA 'Resto da divisão = ', resto
ESCREVA cm, ' cm'
```

O *comando de seleção*, caracterizado pela notação inicial “SE”, permite especificar um desvio condicional de fluxo. Segue o formato:

```
SE EXPR_RELACIONAL
    ENTAO  <COMANDO>
FIM_SE
```

Onde EXPR_RELACIONAL é uma expressão relacional e <COMANDO> refere-se a um ou mais comandos. Por exemplo:

```
SE var1 .M. 2
    ENTAO   ESCREVA var1
FIM_SE
```

```
SE var2 .M. var3
    ENTAO   var1 : var2 + var3
            ESCREVA 'soma = ', var1
FIM_SE
```

O *comando de repetição*, caracterizado pela notação inicial “ENQUANTO”, permite que um determinado comando seja repetido conforme alguma condição. Segue o formato:

```
ENQUANTO EXPR_RELACIONAL
      < COMANDO >
FIM_ENQUANTO
```

Por exemplo:

```
ENQUANTO var1 .M. 10
      var1 := var1 – 1
FIM_ENQUANTO
```

5. Exemplos

A seguir são mostrados dois exemplos de programas escritos na linguagem algo.

Exemplo 1: cálculo de fatorial

```
PROGRAMA fatorial_exemplo
INICIO
        {DECLARACOES }
        INTEIRO argumento, fatorial
        {ALGORITMO }
        { Calcula o fatorial de um número inteiro }
        LEIA argumento
        fatorial := argumento
        SE argumento .I. 0
            ENTAO   fatorial := 1
        FIM_SE
        ENQUANTO argumento .M. 1
                fatorial := fatorial * argumento
                argumento := argumento – 1
        FIM_ENQUANTO
        ESCREVA 'fatorial = ', fatorial
FIM
```

Exemplo 2: ler, armazenar e escrever uma sequencia de números inteiros

```
PROGRAMA leitura de  lista
INICIO
        {DECLARACOES }
        INTEIRO n, i, x, k
        LISTA_INT A[100]
        {ALGORITMO}
            {armazena os dados da lista}
        ESCREVA 'quantos números vai armazenar?'
        LEIA n
        i := 1
        ENQUANTO i .M. n
                LEIA A[i]
                i := i + 1
        FIM_ENQUANTO
           {escreve a lista de numeros}
        ESCREVA 'Numeros armazenados: '
        i := 1
        ENQUANTO i .M. n
              ESCREVA A[i], '  '
              i := i+1
        FIM-ENQUANTO
FIM
```

6. Tarefa

a) Analise léxica: implemente um analisador léxico para a linguagem *Algox*, manualmente como um Autômato Finito Determinístico, ou usando a ferramenta FLEX. Obs.: teste o analisador empregando um arquivo com um programa em *Algox* e gere um arquivo de saída com os tokens detectados.

b) Projete uma estrutura de árvore sintática para *Algox* apropriada para a geração por um analisador sintático.

c) Implemente um analisador sintático descendente para *Algox* (manualmente com base no método descendente recursivo, ou desenvolva o analisador LL(1). Obs.: no teste do analisador utilize o arquivo de saída do item (a) para comprovar se o programa em *Algox* está correto (ou reconhecido).

d) Implemente um analisador sintático ascendente para *Algox* desenvolvendo um analisador LALR(1) em C ou usando a ferramenta YACC. Obs2.: no teste do analisador utilize o arquivo de saída do item (a) para comprovar se o programa em *Algox* está correto (ou reconhecido).

e) Implemente um analisador semântico para a linguagem *Algox*.

f) Implemente uma ferramenta de tabela de símbolos para a linguagem *Algox*.

g) Junte as ferramentas desenvolvidas (em (a), (c) ou (d), (e) e (f)) para implementar o front-end de um gerador de código para *Algox* (síntese das análises)

Obs3.: todos os analisadores devem prever o tratamento de erros.