Linguagem ROTH - Manual
Versão 1.0

Grupo: Guilherme Inácio / Jéssica Soares / Mihael Zamin

Criação de um programa

Os programas da linguagem são constituídos de 4 partes: **declaração de procedimentos**, **declaração de constantes**, **declaração de variáveis** e **corpo do programa**. As três primeiras são opcionais, e a última, o corpo do programa, é obrigatória sendo delimitada por duas palavras-chave: **begin** e **end**. Dentro do corpo do programa pode haver nenhum ou vários comandos, que fazem uma tarefa específica. Um ponto (.) determina o fim do programa. Todos os comandos do programa terminam com (;). Cada parte do programa será vista detalhadamente a seguir.

Sintaxe e estrutura:

```
program identificador_do_programa;

Composição:
   - Seção de declaração de procedimentos
   - Seção de declaração de constantes
   - Seção de declaração variáveis

begin
   - Corpo do programa
end
.
```

A estrutura das seções de declaração são abordadas nas próximas seções deste manual.

Comentários

O comentário é um texto inserido no programa que serve para a documentação do código, facilitando a leitura e compreensão quando o código do programa é lido por outras pessoas. Essas linhas são ignoradas no processo de compilação, não fazendo parte do programa gerado ao final desse processo. Existe apenas o comentário do tipo bloco que no seu conteúdo pode compreender mais de uma linha. Um comentário não tem um limite máximo de tamanho.

Sintaxe:

A sintaxe de comentário de bloco é **/* */**. Ou seja, começam com **/*** e acabam com um ***/**.

Exemplo:

```
/* Isto é um comentário válido */
/* Outro comentário válido
   Segunda linha ---
*/

*/ Esse não é um comentário
```

Tipos, Constantes e Variáveis

Uma constante é como se fosse um apelido para um dado invariável. Podem ser do tipo **array, integer, char, string** ou **real**.

Uma variável é como se fosse um contêiner, mas ela serve para armazenar dados que podem ser mudados, e podem ser do tipo **array, integer, char, string** ou **real**.

Tipos

Tipo **integer**: O integer representa um número do conjunto dos números inteiros, que compreende os números positivos e negativos. Os valores que podem ser representados por um integer devem ser menores que 2^{20} (1.048.576 – valor máximo). Exemplo: 1, 2, 345, 123568.

Tipo **real:** O real representa um número decimal. A faixa de valores que podem ser representados por um real devem ser menores que 4e6.

Tipo **char**: Um char é um caractere alfanumérico único delimitado por aspas simples. Exemplo: ‘a’, ‘B’, ‘1’.

Tipo **string**: Uma string é uma sequência de caracteres alfanuméricos delimitados por aspas duplas. Uma string pode ter até 14 caracteres. Exemplo: “juliano”, “John Doe”, “waffles123”, “123456”.

Tipo **array**: O tipo array representa uma cadeia de elementos de um dos tipos pré-definidos, que são acessados por meio de um índice. É definido por 2 números, como representação das dimensões de uma matriz e o tipo do elemento armazenado, sendo bidimensional, representando uma matriz. Exemplo: array de char de tamanho 3 colunas e duas linhas pode guardar os seguintes dados: [‘a’ ‘b’ ‘C’], array de inteiros de tamanho 5 pode guardar os seguintes dados: [1 2 3 4 5]...

Exemplos de declarações

Uma constante é declarada da seguinte forma:

1) `const identificador = tipo;`

Para mais que uma variável:

2) `const identificador = tipo; identificador = tipo; ...`

Para constantes do tipo array:

3) `const identificador = array[número_de_elementos] of tipo;`

Onde **identificador** é um nome para a constante, para ela poder ser acessada, e tipo é um dos tipos da linguagem descritos anteriormente.

Exemplo:

1) `const pi = real; nomeEmpresa = string;`

As variáveis são declaradas da seguinte forma:

Sintaxe:

1) `declaravariaveis identificador : tipo;`

2) `declaravariaveis identificador1, identificador2, identificador3 : tipo;`

Onde identificador é um nome para a variável, para ela poder ser acessada, e tipo é um dos tipos da linguagem descritos anteriormente.

Exemplo:

1) `declaravariaveis numero : integer;`

2) `declaravariaveis nome1, nome2, nome3 : string;`

Na linha 1 foi declarada uma variável do tipo **integer** chamada numero, e na linha 2 são declaradas 3 variáveis, chamadas nome1, nome2 e nome3, do tipo **string**.

Uma estrutura do tipo **array** é declarada da seguinte forma:

Sintaxe:

```
declaravariaveis identificador : array[número linhas .. numero de colunas] of tipo;
```

Exemplo:

```
/* Um array de integer de 1x2 com 2 elementos */
declaravariaveis arrayDeInteger : array[1..2] of integer;

/* Um array de string com 3x2 com 6 elementos */
declaravariaveis arrayDeString : array[3..2] of string;

/* Acessando o 1º elemento do array */
arrayDeInteger[0..0];
```

Onde identificador é um nome para o array, tipo é um dos tipos da linguagem descritos anteriormente.

Identificadores

Um identificador é um nome de uma variável ou procedimento, que serve para prover acesso aos mesmos, quando se menciona o referido identificador. Ele pode ter no máximo 14 caracteres, podendo ser composto por letras e números, não pode iniciar com número, não pode ter caracteres especiais, exceto o caractere “_”.

Sintaxe:

```
declaravariaveis identificador : tipo;
```

Exemplo:

```
/*Identificador correto*/
declaravariaveis numeroPrimo : integer;

/* Identificador inválido*/
declaravariaveis 1nome234 : string;

/* Identificador inválido */
declaravariaveis 12345 : real;
```

Operadores

Os operadores da linguagem são sinais que fazem operações específicas com o conteúdo de variáveis, ou um valor. Os operadores da linguagem podem ser aritméticos, para efetuar operações aritméticas, lógicos, que comparam uma expressão entre dois valores, e o operador de atribuição, que determina um valor a uma variável. Todos os operadores são binários, ou seja, trabalham com dois operandos, e os operadores de soma (+) e subtração (-) também são unários, podendo trabalhar com somente um operando (-1, +10). Os operadores são os seguintes:

Sintaxe:

Operadores Aritméticos

Soma: +
Subtração: -
Multiplicação: *
Divisão: /

Operadores lógicos

Igual: =
Diferente: <>
Menor: <
Maior: >
Menor e Igual: <=
Maior e Igual: >=
E Lógico: and
Ou Lógico: or

Exemplos:

```
write(2 + 2);          /* Escreve na tela : 4 */
write(3 * 3);          /* Escreve na tela : 9 */
if [ n1 <= 5 ] then /* Se a variável n1 for menor que 5 */
begin
      /* Comandos... */
end
```

Operador de Atribuição

O operador de atribuição é =, e atribui uma variável a um dado ou o conteúdo de outra variável.

```
x = y;      /* Coloca em x o conteúdo da variável y.  */
x = 1234;   /* Coloca em x o valor 1234. */
```

Procedimentos

Procedimentos são sub rotinas formadas por um conjunto de instruções, e podem ou não existir em um programa. A declaração de procedimentos é a primeira seção do programa, podendo ser composta de nenhum, um ou mais procedimentos.
Cada procedimento é composto de definição de **parâmetros de entrada**, declaração de **variáveis internas ao procedimento** e o **corpo** onde estão os comandos de execução, um ponto e vírgula (;) determina o fim do procedimento.

Sintaxe:

```
procedure identificador_do_procedimento(arg1, arg2 ... : tipo; arg3, arg4 ... : tipo ...);
```

Exemplo:

```
procedure Termo(t : string)
/* Declaração de procedimento chamado Termo
O argumento é a variável t
O tipo define qual o tipo de dados será utilizado, nessa caso uma string, cadeia de caracteres */

declaravariaveis i : integer;
/* Logo após o header do procedimento há a declaração de variáveis */

begin
/* Corpo com comandos internos terminados em ; */
end
;
```

Comandos

Comando é uma instrução da linguagem para ser executada pelo programa.

Comandos de chamada de procedimentos

Os comandos de chamada de procedimentos executam procedimentos definidos na seção da declaração de procedimentos.

Sintaxe:

```
chamaprocedure identificador(arg1, arg2, ... argn);
```

Exemplo:

```
chamaprocedure mediaDeNotas(nota1,nota2,nota3);
```

No exemplo acima chama-se o procedimento pré declarado chamado mediaDeNotas que utiliza os argumentos nota1, nota2, nota3 para execução de seus comandos.

Comandos de entrada e saída

Os comandos de entrada e saída servem para receber e mostrar dados ao usuário. O comando de saída, **write**, pode exibir o conteúdo de uma variável, string ou o resultado de uma expressão, ou uma literal, que é uma seqüência de caracteres. Um literal pode ter até 32 caracteres. O comando de leitura, **read**, recebe um valor informado pelo usuário, e o coloca em uma variável. O tipo de valor lido deve ser compatível com o tipo da variável.

Sintaxe:

```
write(identificador, expressão ou string);
read(nome da variável que receberá o valor);
```

Exemplo:

```
write($O rato roeu a roupa do Rei de Roma$);
write(2 + 2);            /* Escreve 4 */
read(numero);
```

Comandos de repetição

Estrutura de decisão – if then...else

A estrutura de decisão **if then...else** permite que seja executado um bloco de comandos a partir da avaliação de uma expressão. Se o resultado dessa expressão for verdadeiro, é executado o bloco de comandos do **if**. Se a expressão é falsa, é executado o bloco de comandos de **else,** sendo a parte else opcional.

Sintaxe:

```
if [ expressão ] then
      begin
            /* Bloco de comandos do if...  */
      end
 /* opcional */
else
      begin
            /* Bloco de comandos do else... */
      end
```

Exemplo:

```
if [ num < 0 ] then
      begin
            write($Numero negativo$);
      end
else
      begin
            write($Numero positivo$);
      end
```

No exemplo acima, é testado se o valor da variável num, que é do tipo integer, é menor que zero. Se isso é verdadeiro, é escrito na tela Número negativo. Senão é escrito na tela Número positivo. Somente um comando de escrita irá ser executado.

Estrutura de repetição for

A estrutura de repetição for repete enquanto uma expressão for verdadeira por um número pré definido de vezes. O número de iterações deve ser definido na declaração do laço for.

Sintaxe:

```
for [ identificador = expressão1 ] to [ expressão2 ] do
begin
      /* Comandos... */
end
```

Onde identificador é uma variável de controle do laço, que irá controlar o número de iterações. A expressão1 é uma expressão que determina o valor inicial, e expressão2 determina o valor final. Os comandos do bloco são executados, e a variável de controle é incrementada pelo laço, até chegar ao valor final.

Exemplo:

```
declaravariaveis i : integer;
i := 0;

for [i = 1] to [10] do
      begin
            write(i);
      end
```

No exemplo acima, foi criado uma variável do tipo integer, que recebe o valor 0. No for, a variável i é a variável de controle. O laço executará de 1 até chegar em 10 (enquanto 1 for menor que 10), imprimindo na tela o valor atual da variável de controle.

Estrutura de repetição while

A estrutura de repetição while repete comandos enquanto uma expressão for verdadeira por um número não determinado de vezes. O número de iterações depende da mudança do resultado lógico da expressão testada.

Sintaxe:

```
while [ expressão ] do
begin
      /* Comandos... */
end
```

Onde expressão é uma verificação lógica, que definirá quando o laço deve parar.

Exemplo:

```
declaravariaveis i : integer;

while [ i < maxValor ] do
Begin
      /* Comandos... */
      i = i + varAcrescimo;
end
```

No exemplo acima o é testado se a variável i atinge o valor máximo (maxValor), e executa os comandos atualizando o novo valor de i para o próximo laço.

Estrutura de repetição repeat

A estrutura de repetição repeat executa laços de comando até que a expressão em until seja verificada como falsa.

```
repeat
      /* Comandos... */
until [ expressão ]
```

Onde expressão é uma verificação lógica, que, se falsa, definirá quando o laço deve parar.

Exemplo:

```
declaravariaveis i : integer;

repeat
      write($Verdadeiro$);
      i = i + 1;
until [ i < 500 ]
```

O exemplo acima irá exibir a expressão Verdadeiro até que o valor de i chegue em 500.

Exemplo de programa

```
program identificador_do_programa;
procedure iguais(a, b : integer)
declaravariaveis texto : string;
begin
texto = “São iguais.”;
if [a = b] then
      begin
            write(texto);
      end
else
      begin
            write($Não são iguais.$);
      end
end
;
```

```
const mensagem_usuario = string;
declaravariaveis num1, num2 : integer;

begin
      mensagem_usuario = “Digite um número:”;
      write(mensagem_usuario);
      read(num1);
      write(mensagem_usuario);
      read(num2);
      chamaprocedure iguais(num1, num2);
end
.
```

O exemplo acima é um programa simples que se utiliza de várias estruturas da linguagem com o objetivo de mostrar ao usuário como um programa completo se parece.

Erros léxicos

Os erros léxicos são identificados durante a análise léxica. A análise ocorre em 3 etapas: a primeira remove todos os caracteres de quebra de linha, na segunda o scanning remove todos os comentários. Por fim, na terceira, agrupa os caracteres em tokens. Os tokens representam um tipo de unidade léxica, como: os identificadores, números, palavras, símbolos e etc.

São considerados erros léxicos da linguagem a utilização de símbolos não existentes na linguagem, atribuição de valores superiores ao limite superior, em variáveis do tipo integer e real, não colocação da tag de fechamento de comentário de bloco (*/), não colocação da tag de fechamento de literal($) e literais com mais de 32 caracteres e identificadores com mais de 14 caracteres.

Exemplo de erro léxico:

```
fi [a=b] then /* Espera-se encontrar if [a=b] then */
```

Não é possível ao analisador léxico determinar se foi um erro de digitação do identificador if.
Nesse caso existem a estratégia é usar um panic-mode, que apaga os caracteres da sequência até um token válido. Assim será possível manter o processamento.
Os erros podem ser vistos num relatório gerado após a compilação, devendo possibilitar a rápida identificação para correção.