UNIVERSIDADE DO EXTREMO SUL CATARINENSE - UNESC

CURSO DE CIÊNCIAS DA COMPUTAÇÃO

CAROLINE SALIB CANTO

HELDER ROCHA DA SILVA

MANUAL DA LINGUAGEM UNSAUBER

CRICIÚMA

2016

CAROLINE SALIB CANTO

HELDER ROCHA DA SILVA

MANUAL DA LINGUAGEM UNSAUBER

Trabalho solicitado pela disciplina de Compiladores, do curso de Ciências da Computação da Universidade do Extremo Sul Catarinense, UNESC.

Orientadora: Profª Christine Vieira

CRICIÚMA

2016

SUMÁRIO

MANUAL DA LINGUAGEM

1. LINGUAGEM

Uma linguagem de programação é composta por um conjunto de símbolos e regras de sintaxe, que permitem a construção de sentenças para descrever ações que são executadas pelo computador. Este manual consiste na descrição dos símbolos utilizados pela linguagem Unsauber e suas regras de sintaxe.

2. ESTRUTURA DO PROGRAMA

2.1. Inicio do programa

Para iniciar um programa, deve-se usar a palavra reservada "`program`", e na sequencia o nome do programa (sem espaços ou caracteres especiais), finalizando com um ponto-e-virgula.

```
program NOME_DO_PROGRAMA;
```

2.2. Label

Label é o endereço de destino após a utilização do comando "`goto`", o mesmo é iniciado pelo prefixo "`label`" e finalizado com ponto-e-vírgula após o término da declaração.

```
label a;
```

2.3. Constantes

Na programação, Chamamos de constantes algo que nunca irá mudar o seu valor. Por exemplo, ao resolver um problema matemático que envolve o número π (PI), poderíamos declarar uma constante "PI" para utilizarmos na conta matemática. Dessa forma, o código ficaria mais simples de entender, sem "números mágicos" e inclusive mais seguro, pois durante o decorrer do código o "PI" nunca nunca poderá ser falsificado. Para declaração de uma constante deve-se usar o prefixo "`const`".

```
const DIVIDENDO = 2;
```

Obs.: Nesta linguagem, a constante só poderá receber um valor do tipo inteiro.

2.4. Variáveis

Variáveis são recursos utilizado para escrever e ler dados da memória do computador, que é na verdade um espaço de memória que nomeamos. Como as variáveis são utilizadas para armazenar dados, podemos dizer que uma variável chamada "nome" irá

receber o nome de uma pessoa. Nesta linguagem de programação podem assumir o valor int ou array de ints.
Para a declaração das variáveis devemos iniciar com a palavra "`var`" seguida pelo nome da variável e por fim o tipo, exemplo:

```
var variavel1 = int;
var variavel1 = array [integer .. integer];
```

Obs.: Os identificadores das variáveis só podem conter letras ou números, sem o uso de caractéres especiais.

2.5. Procedures

É possível dividir a lógica de programas complexos em programas menores, depois juntá-los à fim de compor o programa final. Facilitando a construção de grandes programas, dividindo em pequenas etapas as rotinas que devem ser executadas, reaproveitando as mesmas, utilizadas quantas vezes necessitar não tendo que repedir as programações das mesmas instruções do código. Esta prática é conhecida como declaração de procedimento, neste caso procedure.
Para a declaração de uma procedure deve-se simplesmente declarar seu nome, seguido de parâmetros (opcional) do tipo `int`, especificado após o dois-pontos. Utilize dois-pontos para indicar o inicio do bloco, e após definir o bloco de código da procedure, deve-se finalizar a mesma utilizando ponto-e-vírgula.

```
procedure soma (valor1 : int, valor2 : int) :
   // bloco
;
```

2.6. Corpo do programa

O corpo do programa é parte obrigatória, o começo é determinado por "`begin`" e o final, por "`end`", exemplo:

```
begin

   //comandos
end
```

3. BLOCO

3.1. Hierarquia

Para que o bloco funcione corretamente, o mesmo deve respeitar a hierarquia determinada pela linguagem. Esta hierarquia deve conter os elementos na seguinte sequencia:

- Label (Opcional)
- Constantes (Opcional)

- Variáveis (Opcional)
- Procedures (Opcional)
    - O bloco dentro da procedure deve seguir a mesma hierarquia proposta neste tópico.
- Corpo (Obrigatório)

Exemplo:

```
label a;
const nome_constante = integer;
var variavel1 = int;

procedure soma (valor1 : int, valor2 : int) :
   // bloco
;

begin
   //comandos
end
```

4. COMANDOS

4.1. Call

Utiliza-se o comando call quando for necessário fazer uma chamada de procedure, exemplo:

```
procedure soma (valor1 : int, valor2 : int) :
   // bloco
;

begin
   call soma(1, 2)
end
```

4.2. GoTo

O comando **goto** (do inglês “ir para”) é utilizado para fazer saltos de instruções. Para usar esse comando, deve ser declarado um **label** com o nome do destino, e usar o **goto** para apontar para este destino, exemplo:

```
label destino

// outras instruções

goto destino
```

4.3. ReadIn

O comando **readln** é um comando de entrada que utiliza-se a fim de obter dados digitados pelo usuário, exemplo:

```
readln (nome_variavel)
```

Obs.: A variável com o nome de “nome_variavel” será o lugar onde o programa armazenará o valor recebido.

4.4. WriteIn

O comando writeln é um comando de saída que utiliza-se a fim de enviar mensagens ao usuário em questão, exemplo:

```
writeln ('Olá, qual seu nome?')
```

4.5. If e Else

Através do comando **if** (se) e **else** (se não), é possível condicionar o código do programa, podendo então verificar se determinada condição é verdadeira ou falsa, e tomar a devida providencia para cada uma das condições necessárias, exemplo:

```
 if true then
    // comandos para condição verdadeira
 else
    // comandos para condição falsa
```

Obs.: Na sessão de “expressões” será visto como usar expressões para condicionar o código de forma mais específica.

4.6. Case

O comando **case** é importante para a estruturação de um programa que possua diversas opções de execução, tornando-o bem legível e estruturado, evitando o uso do **if** de forma repetida, exemplo:

```
case 1 of
1 : begin
      //comando quando for 1
   end;
2 : begin
      //comando quando for 2
   end;
3 : begin
      //comando quando for 3
```

```
   end
end
```

4.7. While

O loop while é um comando que permite o código ser executado repetidamente através de uma expressão verdadeira, exemplo:

```
while true do
begin
   //comando a ser executado
end
```

4.8. Repeat

O comando **repeat** (do inglês “repita”) **until** (até) é similar ao comando **while**, a diferença principal entre os dois é que, no comando **repeat**, a sequência de comandos a repetir é executada sempre pelo menos uma vez, enquanto que no comando **while** o comando a repetir pode não ser nunca executado, exemplo:

```
repeat
begin
   //comando a ser executado
end
until true
```

4.9. For

O comando for, diferentemente dos comandos de repetição **repeat** e **while**, permite que uma sequência de comandos seja executada um número definido de vezes A variável usada para contador já recebe um valor inicial e é incrementada pelo comando, exemplo:

```
for contador := 1 to 10 do
   writeln (contador)
end
```

O exemplo acima mostra ao usuário todos os números de “um” até “dez”.

5. OPERADORES

Os operadores são meios pelo qual podemos incrementar, decrementar, comparar e avaliar o conteúdo da variável, ou um valor. Os operadores desta linguagem podem ser **aritméticos**, **comparativos** e **lógicos**. Os operadores **aritméticos** são utilizados para obter resultados numéricos, neste caso pode ser adição, subtração, multiplicação e divisão. **Operadores comparativos** são utilizados para comparar uma

variável ou valor, nesta linguagem, somente inteiros. Como o igual, diferente, menor, maior, menor ou igual a e maior ou igual a. E, os **operadores lógicos** sempre retornam valores lógicos, ou seja, verdadeiro ou falso. Que são representados por E (and) e OU (or).

Sintaxe:

5.1. Operadores aritiméticos

Adição	**+**
Subtração	**-**
Multiplicação	*****
Divisão	**/**

Exemplo:

```
writeln(1 + 2 - 3) //Escreve o resultado da expressão: 0
writeln(2 * 2 / 2) //Escreve o resultado da expressão: 2
```

5.2. Operadores comparativos

Igual	**=**
Diferente	**<>**
Menor	**<**
Maior	**>**
Menor ou igual a	**<=**
Maior ou igual a	**>=**

Exemplo:

```
if 1 < 2 then
   writeln("um é menor que dois")
else
   writeln("um não é menor que dois")
```

5.3. Operadores lógicos

Ou	**not**
E	**and**

Exemplo:

```
if not(1 < 2) then
   writeln("um não é menor que dois")
else
   writeln("um é menor que dois")
```

5.4. Operadores de atribuição

Atribuição de valor ou variável	`:=`

Exemplo:

```
a := b;   // Adiciona na variável "a" o conteúdo da variável "b".
a := 123; // Adiciona na variável "a" o valor 123.
```

6. COMENTÁRIOS

Podem ser inseridos textos no decorrer do programa para descrever uma rotina e documentar o que está sendo declarado naquele comando, é muito utilizado para facilitar a compreensão do código do programa quando é lido por outras pessoas ou até mesmo pela própria pessoa que o digitou. Para que este texto não seja entendido como parte do código, são utilizados dois comandos: um para comentário de linha (//) e outro para comentário de bloco (/* e */).

O comentário de linha se estende por apenas uma linha, enquanto o de bloco pode ser estender por mais de uma linha. Um comentário não tem um limite máximo de tamanho.

6.1. Comentário de linha

A declaração do comentário de linha é feito por duas barras // seguidas do texto do comentário.

Exemplo:

```
// Este é um comentário de linha.
```

6.2. Comentário de bloco

A declaração do comentário de bloco é iniciada por uma barra e um asterisco (/*), seguido do texto e finalizando com um asterisco e uma barra (*/).

Exemplo:

```
/* Este é um
comentário
de bloco
*/
```

7. ERROS LÉXICOS

São considerados erros léxicos da linguagem Unsauber o uso de caracteres (ou símbolos) inválidos, que não foram pré-determinados no manual da linguagem. Também são erros léxicos o mau uso da sintaxe estabelecida, como por exemplo, a ausência de parte de um comando ou expressão.