Modularização

Um matemático uma vez disse que um grande problema se resolve dividindo-o em pequenas partes e resolvendo tais partes em separado. Estes dizeres servem também para a construção de programas. Os profissionais de informática quando necessitam construir um grande sistema, o fazem, dividindo tal programa em partes, sendo então desenvolvida cada parte em separado, mais tarde, tais partes serão acopladas para formar o sistema. Estas partes são conhecidas por vários nomes. Nós adotaremos uma destas nomenclaturas: sub-Rotinas.

Podemos dar um conceito simples de sub-Rotina dizendo ser um pedaço de código computacional que executa uma Função bem definida, sendo que esta sub-Rotina pode ser utilizadas várias vezes no algoritmo.

Iremos tratar de dois tipos de sub-Rotinas: "PROCEDURE" e FUNÇÃO.

As **PROCEDURES** e as **FUNÇÕES** são módulos de programação que servem basicamente a três objetivos:

1) Evitar que, uma certa seqüência de comandos necessária em vários locais de um algoritmo tenha que ser escrita repetidamente nestes locais.
2) Dividir e estruturar um algoritmo em partes fechadas e logicamente coerentes.
3) Aumentar a legibilidade de um algoritmo.

PROCEDURE (Procedimento)

Uma "PROCEDURE", é um tipo de sub-Rotina que é ativada através da colocação de seu Nome em alguma parte do programa. Desta forma, assim que o Nome de uma "PROCEDURE" é encontrado, ocorre um desvio no programa, para que os comandos da sub-Rotina sejam executados. Ao término da sub-Rotina, a execução retornará ao ponto subseqüente a chamada da "PROCEDURE".

Declaração em Português estruturado:

```
Procedure <NOME> (var: tipo) {Início do bloco Lógico}
        {Conjunto de ações}

Fim_Procedure {Fim do bloco lógico}
```

Onde:

Procedure	é uma palavra-chave;
NOME	é o nome dado à sub-rotina;
Var: tipo	é a variável e o tipo da mesma que está sendo passada como parâmetro
Conjunto de ações	é a lista dos objetos que serão substituídos por outros objetos, fornecidos quando da Chamada da sub-rotina.

Declaração na linguagem C:

```
void <NOME> (tipo var)
{ // início do bloco
        {Conjunto de ações}
  } // fim do bloco
```

Onde:

void	é uma palavra-chave que indica que é uma procedure;
NOME	é o nome dado à sub-rotina;
Var: tipo	é a variável e o tipo da mesma que está sendo passada como parâmetro
Conjunto de ações	é a lista dos objetos que serão substituídos por outros objetos, fornecidos quando da Chamada da sub-rotina.

Exemplo 1:

```
Programa Teste_Procedure

VAR
    num, N :  inteiro

{Declaração da Procedure)

Procedure  EscreveNoVideo
    PARA (num ←1, num<= N , num ← num+1)
        ESCREVA (num)
    FIM PARA
Fim-Procedure

Inicio
    LEIA(N)
    EscreveNoVideo
    ESCREVA ('fim')
Fim
```

```c
#include <stdio.h>
#include <stdlib.h>

int num, N;

void EscreveNoVideo ( )
{
    for (num = 1; num <= N; num++)
      printf ("%d\n", num);
}

main ()
{
      printf("Digite um numero\n");
      scanf ("%d", &N);
      EscreveNoVideo( );
      printf ("FIM");
      system ("pause");
}
```

Exemplo: O programa cria um procedimento chamado EscreveNoVideo, depois no programa principal se digita um número e se executa o procedimento EscreveNoVideo que escreve os números de 1 até o número digitado. (exemplo1.c)

Exemplo 2:

```
Programa  Exemplo_Procedure

Procedure TROCA (A,B: inteiro)
Var
        AUX : inteiro
Inicio
        AUX ← A
        A ← B
        B ← AUX
        Escrever (A, B)
Fim_procedure
Var
        L,M,N : inteiro

Inicio
         Ler (L,M,N)
        TROCA (L,M)
Fim             {Fim do algoritmo}
```

```c
#include <stdio.h>
#include <stdlib.h>

void Troca (int A, int B)
{
      int aux;

      aux = A;
      A = B;
      B = aux;
      printf ("\n\n%d  %d\n", A,B);
}
main()
{
      int L, M;

      L = 2;
      M = 5;
      printf ("%d   %d\n\n", L,M);
      Troca (L, M);
      system ("pause");
}
```

Exemplo: O programa cria um procedimento chamado Troca, depois no programa principal se declara duas variáveis inteiras L e M com os valores 2 e 5 respectivamente. Ao chamar o procedimento Troca o mesmo faz a troca dos valores das variáveis L e M e imprime. (exemplo2.c)

No exemplo acima, cada vez que a sub-rotina TROCA for ativada, os comandos, dentro dela são executados tendo em vista os valores contidos nos parâmetros atuais e, em seguida, a seqüência do algoritmo retorna ao comando imediatamente seguinte ao da ativação.

Criar parâmetros significa definir variáveis que servem para transformar informações especificas que funcionam como entrada para o conjunto de ações do módulo que está sendo ativado.

Os parâmetros de uma sub-rotina classificam-se em:

- Parâmetros de entrada - são aqueles que têm seus valores estabelecidos fora da sub-rotina e não podem ser modificados dentro da sub-rotina.
- Parâmetros de saída - são aqueles que têm seus valores estabelecidos dentro da sub-rotina.

Variáveis Locais e Globais

Variáveis globais: são variáveis definidas logo após o comando VAR do programa principal, sendo, desta forma, visíveis em qualquer parte do programa.

Na linguagem C as variáveis globais são declaradas fora das sub-rotinas e fora do programa principal (main).

Exemplo 3:

```
Programa Teste

VAR
      Num :  INTEIRO{variável global}

Procedure SetaNumero
      LEIA (Num)
Fim-Procedure

Inicio
      SetaNumero
      ESCREVA (Num)
Fim
```

```
#include <stdio.h>
#include <stdlib.h>

int Num; // variável global

void SetaNumero( )
{
      printf ("Digite um numero: ");
      scanf("%d", &Num);
}
main()
{
  SetaNumero( );
  printf ("Numero digitado foi: %d\n", Num);
  system ("pause");
}
```

Exemplo: O programa utiliza de um procedimento, porém com a declaração de uma variável global. (exemplo3.c)

No Exemplo acima, a variável "Num", por ser definida como global, pode ser manipulada dentro de qualquer ponto do programa, sendo que qualquer mudança no seu conteúdo será visível nas demais partes da Rotina.

Variáveis locais: são variáveis que são declaradas dentro de uma sub-Rotina, sendo que as mesmas só podem ser manipuladas dentro da sub-Rotina que as declarou, não sendo visíveis em nenhuma outra parte do programa.

Exemplo:

```
Programa Testa_variável

Procedure EscreveNoVídeo
   Var
      Num, N: INTEIRO
  Inicio
     LEIA (N)
     PARA (Num DE 1 ATE N passo 1) FAÇA
           ESCREVA (Num)
     FIM PARA
Fim-Procedure

Inicio
      EscreveNoVídeo
Fim
```

```
#include <stdio.h>
#include <stdlib.h>
void EscreveNoVideo( )
{
      int Num, N;
      printf ("Digite um numero: ");
      scanf("%d", &N);
      for (Num = 1; Num <= N; Num++)
            printf ("%d  ", Num);
}
main()
{
      EscreveNoVideo( );
      system ("pause");
}
```

Exemplo: O programa utiliza de um procedimento, porém com a declaração de variáveis locais. (exemplo4.c)

FUNÇÃO

Uma sub-Rotina do tipo “FUNÇÃO” possui as mesmas características de uma “PROCEDURE” no que se refere a passagem de parâmetros, variáveis globais e locais, mas possui uma importante diferença, que é o retorno de um valor ao término de sua execução, ou seja, uma FUNÇÃO **sempre** deverá retornar um valor ao algoritmo que a chamou.

Na definição de uma “FUNÇÃO”, deverá ser informado qual o tipo do valor retornado, sendo que poderá ser usado, nesta definição, tanto tipos pré-definidos da linguagem, como tipos definidos pelo usuário. Somente não poderão ser retornados tipos ARRAY (Vetores e matrizes) e RECORD (registros), justamente por serem tipos que definem variáveis que armazenam mais de um valor.

Declaração:

```
Função Tipo NOME (lista-de-parâmetros-formais)
        {declaração dos objetos locais à função}
        {comandos da função}
Fim_ Função
```

Onde:

Função	é uma palavra-chave;
Tipo	é o tipo do valor que será retornado;
NOME	é o nome dado à função;
Lista-de-parâmetros-formais	é a lista dos objetivos que serão substituídos por outros objetos, fornecidos quando da Chamada da função.

Para informar qual o valor deve ser retornado deve ser colocada, em algum ponto do código da “FUNÇÃO”, uma linha com a seguinte sintaxe:

<Nome da FUNÇÃO > ← < o valor a ser retornado>

A forma geral para ativação de uma função:
NOME (lista-de-parâmetros-atuais)

Onde:

NOME	é o nome dado à função;
Lista-de-parâmetros-atuais	é a lista de objetos que substituirão os parâmetros formais durante a execução da função. Os parâmetros atuais devem concordar em número, ordem e tipo com os parâmetros formais.

Declaração na linguagem C:

```
Tipo  <NOME> (tipo var)
{ // início do bloco
        {Conjunto de ações}
  } // fim do bloco
```

Onde:

Tipo	é o tipo do valor que será retornado;
NOME	é o nome dado à função;
Tipo var	são as variáveis e o tipo das mesmas que estão sendo passadas como parâmetros
Lista-de-parâmetros-formais	é a lista dos objetivos que serão substituídos por outros objetos, fornecidos quando da Chamada da função.

Na linguagem C para informar qual o valor deve ser retornado este deve ser colocado com a seguinte sintaxe:

return (valor/variável a ser retornada);

A forma geral para ativação de uma função:

Variável ← NOME (Lista-de-parâmetros-atuais)

Onde:

Variável	é a variável que recebe o valor da função
NOME	é o nome dado à função

Lista-de-parâmetros-atuais	é a lista de objetos que substituirão os parâmetros formais durante a execução da função. Os parâmetros atuais devem concordar em número, ordem e tipo com os parâmetros formais.

Exemplo:

```
Programa Exemplo_Função

{Definição da função}

Inteiro SOMA (V1, V2: Inteiro)
 Var
    S: inteiro
Inicio
     S ← V1 + V2
     retorna (S)
Fim_Função

Var
        K: Inteiro

Inicio
        K ← SOMA (2,3)
        Escreva (K)
Fim
```

```c
#include <stdio.h>
#include <stdlib.h>

// declaração da função
int Soma (int V1, int V2)
{
        int S;
        S = V1 + V2;
        return (S);
}

main()
{
        int K;
        K = Soma (2,3);
        printf ("Resultado: %d\n", K);
}
```

Exemplo: O programa utiliza uma função que retorna a soma de dois nºs inteiros passados por parâmetro. (exemplo5.c)