MODULARIZAÇÃO – CONCEITOS GERAIS

Um programa pode ser construído como um conjunto de SUBROTINAS.
Cada subrotina deve ser elaborada pensando-se na possibilidade de sua reutilização em vários programas.
As subrotinas podem ou não ter parâmetros, ou seja, pode haver informações que sejam enviadas a ela para que a tarefa seja realizada.
Para executar uma subrotina basta ativá-la em alguma linha de código do programa (ponto de chamada ou ponto de ativação).

– TIPOS DE SUBROTINAS:
FUNÇÃO (modelo clássico), retorna 1 único valor
PROCEDIMENTO (modelo clássico), não retorna nada
PROCEDIMENTO COM PARÂMETROS POR REFERÊNCIA, retorna 2 ou mais valores

– ESCOPO DAS VARIÁVEIS:
GLOBAL
LOCAL <- priorizar !!! Mais segurança e trabalha em conjunto com os parâmetros das subrotinas.

– PASSAGEM DE PARÂMETROS:
POR VALOR, repassa à subrotina CÓPIA do conteúdo de variável ou valor constante definido
POR REFERÊNCIA, faz com que a subrotina acesse a variável original, mesmo ela estando declarada em outro escopo. Em pseudocódigo usa-se a palavra reservada VAR e na linguagem C++ usa-se o operador de referência & no protótipo da subrotina.

MODULARIZAÇÃO – FUNÇÕES:

***FORMA GERAL de uma função*:**

```
FUNÇÃO nome_da_função ( p1: TIPO, p2, p3: TIPO, ... pn: TIPO ) : TIPO
VARIÁVEIS
  { declaração das variáveis locais }
INÍCIO
  <comandos>
  nome_da_função <- valor
FIM
```

EXEMPLO:

```
FUNÇÃO leiaNumInteiroPositivo () : INTEIRO
VARIÁVEIS
  INTEIRO n
INÍCIO
ESCREVA("Valor: ")
REPITA
  LEIA(n)
ATÉ (n>0)
leiaNumInteiroPositivo <- n
FIM
```

```
FUNÇÃO leiaNumInteiroIntervalo (inf, sup: INTEIRO) : INTEIRO
VARIÁVEIS
  INTEIRO n
INÍCIO
ESCREVA("Valor: ")
REPITA
  LEIA(n)
ATÉ (n>=inf E n<=sup)
leiaNumInteiroIntervalo <- n
FIM
```

EM LINGUAGEM C++:

```
tipoDoRetorno nome_funcao (tipo p1, tipo p2, ...) {
  <comandos>
  return valor;
}
```

```
int leiaNumInteiroPositivo ( ) {
int n;
cout <<"Valor: ";
do{
  cin >> n;
} while (n<=0);
return n;
}
```

```
int leiaNumInteiroIntervalo (int inf, int sup)
{
int n;
cout <<"Valor: ";
do{
  cin >> n;
} while (n<inf || n>sup);
return n;
}
```

MODULARIZAÇÃO – PROCEDIMENTOS:

***FORMA GERAL de um procedimento (CLASSICO)*:**

PROCEDIMENTO nome_do_procedimento **(** p1: TIPO, ... pn: TIPO **)**
VARIÁVEIS
{ declaração das variáveis locais }
INÍCIO
<comandos>
FIM

EXEMPLO:

```
PROCEDIMENTO MostraMsg ( codigoDia: INTEIRO )
INÍCIO
ESCOLHA codigoDia
   CASO 0: ESCREVA (“domingo”)
   CASO 1: ESCREVA (“segunda”)
   CASO 2: ESCREVA (“terça”)
   CASO 3: ESCREVA (“quarta”)
   CASO 4: ESCREVA (“quinta”)
   CASO 5: ESCREVA (“sexta”)
   CASO 6: ESCREVA (“sábado”)
FIMESCOLHA
FIM
```

EM LINGUAGEM C++:

```
void nome_procedimento (tipo p1, tipo p2, ...) {
        <comandos>
}
```

```
void mostraMsg ( int codigoDia ) {
  switch (codigoDia) {
     case 0: cout <<“domingo”<< endl; break;
     case 1: cout <<“segunda”<< endl; break;
     case 2: cout <<“terça”<< endl; break;
     case 3: cout <<“quarta”<< endl; break;
     case 4: cout <<“quinta”<< endl; break;
     case 5: cout <<“sexta”<< endl; break;
     case 6: cout <<“sábado”<< endl;
  }
}
```

MODULARIZAÇÃO – PROCEDIMENTOS COM PARÂMETROS POR REFERÊNCIA:

FORMA GERAL de um procedimento com parâmetros por referência (VAR):

```
PROCEDIMENTO nome_do_procedimento ( VAR p1: TIPO, ... pn: TIPO )
VARIÁVEIS
{ declaração das variáveis locais }
INÍCIO
<comandos>
FIM
```

EXEMPLO:

```
PROCEDIMENTO leiaCoordTela ( VAR x, y: INTEIRO )
INÍCIO
    x= leiaNumInteiroPositivo()
    y= leiaNumInteiroPositivo()
FIM
```

CHAMADA:

```
ALGORITMOS XXX
   x1, y1, x2, y2: INTEIRO
   ...
   leiaCoordTela ( x1, y1 );
   ...
}
```

EM LINGUAGEM C++:

```
void nome_procedimento (tipo & p1, tipo & p2, ...) {
        <comandos>
}
```

CODIGO:

```
void leiaCoordTela ( int &x, int &y ) { // OS PARÂMETROS SÃO REFERÊNCIAS
    x= leiaNumInteiroPositivo();
    y= leiaNumInteiroPositivo();
}
```

CHAMADA:

```
int main() {
   int x1, y1, x2, y2;
   ...
   leiaCoordTela ( x1, y1 ); // CHAMA A SUBROTINA ACIMA PARA 1º PONTO
   gotoxy(x1,y1);
   ...
   leiaCoordTela ( x2, y2 ); // CHAMA A SUBROTINA ACIMA PARA 2º PONTO
   gotoxy(x2,y2);
   ...

}
```