Capítulo 6 – Abstrações de comandos

Blocos de comandos

Um bloco de comandos pode ser tomado como um trecho de algoritmo que constitui, por si só, uma identidade funcional. Um bloco deve exprimir uma ação, ou o conjunto de instruções primitivas que a descreve; além disso, poderá conter, dependendo da necessidade e situação, definições, comandos, ou outros blocos mais internos.

Um bloco, em geral, é delimitado por indicadores de início e fim, podendo pertencer, ou não, a estruturas de controle. Dentro desses limites, os dados definidos localmente passam a existir, e podem ser manipulados até o término da execução dos comandos desse bloco. Esses dados serão chamados de locais, os dados que pertencerem a blocos mais externos, como o do algoritmo completo, serão chamados de globais. Assim, a definição de blocos dentro de outros, com seus respectivos limites, estabelece um escopo no qual os dados têm o seu tempo de vida definido. Tipos especiais de blocos podem permitir a alocação dinâmica da memória, com a vantagem de que só é reservado para uso o espaço para as necessidades de um determinado bloco, sendo liberado em seguida, após sua execução.

O uso de blocos permite construir algoritmos melhor estruturados, facilitar a compreensão e a manutenção.

Chama-se de abstração de comandos às formas agrupadas, em blocos, de um conjunto de entidades e instruções, responsáveis por executar uma ação bem definida, podendo, ou não, repetir-se em diferentes pontos de um algoritmo.

Esse conceito de abstração de comandos pode ser aplicado a vários tipos especiais :

- macros
- funções
- procedimentos
- construtores
- módulos
- métodos
- operadores.

O uso destas abstrações pode trazer várias vantagens :

- facilitar a confecção, teste e manutenção de algoritmos;
- melhorar a legibilidade do algoritmo;
- aumentar a portabilidade e eficiência do algoritmo;
- permitir o compartilhamento entre algoritmos.

Modularização

Chama-se *modularização* à possibilidade de se compor uma ação a partir de outras ainda mais primitivas, agrupando-as em unidades sintaticamente fechadas e logicamente relacionadas Isso pode trazer uma série de vantagens a um algoritmo:

- aumentar a legibilidade, a eficiência e portabilidade;
- facilitar a execução de testes e sua manutenção;
- permitir o isolamento de erros e sua correção;
- permitir compartilhamento e desenvolvimento paralelo.

A modularização pode tornar uma solução bem mais fácil de ser alcançada, supondo que as partes menores possuam uma complexidade menor que a do problema original. Quanto mais independentes estas partes (os módulos), mais fácil é o desenvolvimento de cada uma delas, podendo, inclusive, ficar a cargo de pessoas diferentes para fazê-lo.

Ao grau de interdependência das partes chama-se *acoplamento*, e quanto menor, mais fácil a manutenção. Serve como fator de medida de qualidade. Alguns fatores podem influenciá-lo:

- quantidade de dados e comandos;
- complexidade das partes;
- tipo de acoplamento: por dados, por controle etc.

Ao grau de relacionamento entre as entidades e ações agrupadas em um bloco chama-se *coesão*, e quanto maior, melhor a estrutura. Esse também é um fator de qualidade. Há vários tipos de coesões:

- *coincidência* : não existe coesão;
- *lógica* : em um classe de operações;
- *temporal* : relacionadas pela execução;
- *procedimental* : execução seqüencial de comandos;
- *comunicação* : operações sobre os mesmos dados;
- *sequencial* : a saída de um bloco é a entrada de outro;
- *funcional* : para os mesmos dados, mesmo objetivo.

A coesão pode ser avaliada pela análise de algumas propriedades:

- *necessidade* : cada módulo contribui com uma parte da solução;
- *suficiência* : somente o conjunto de todos os módulos resolve por completo o problema;
- *integrabilidade* : os módulos permitem compor suas implementa-ções;
- *minimalidade* : não há outro conjunto mínimo que satisfaça a mesma solução;
- *ortogonalidade* : cada módulo executa apenas a sua função e não se superpõe a nenhum outro;
- *completude* : a especificação de cada módulo e sua interface estão completos;
- *viabilidade* : cada módulo do conjunto solução é passível de implementação.

Na prática, o ideal é que cada módulo seja relativamente simples, responsável pela consecução de um objetivo bem definido e executado o mais independentemente possível de outros módulos. Uma outra característica importante é apresentar apenas um ponto de entrada e saída, pois isto reflete uma boa consistência interna e definição funcional, sendo assim, favorecerá o seu entendimento, testes e manutenção.

A comunicação entre módulos deve se dar através de interfaces bem definidas, capazes de transmitir dados de maneira eficiente, reforçando uma boa escolha de suas definições.

Depuração e validação

Não basta ao processo de desenvolvimento apenas produzir soluções modulares. É necessário que cada módulo funcione corretamente.

À atividade de teste e verificação de um módulo chama-se *depuração*. Os testes devem ser previstos durante a etapa de concepção do próprio módulo e deve anteceder a sua integração com os outros.

Os *testes de integração* têm o objetivo de verificar o funcionamento harmônico dos módulos e garantir que a comunicação entre eles se dê conforme o projetado.

A partir disso, deve-se submeter o conjunto aos *testes de validação*. Nessa etapa os dados conhecidos do problema serão submetidos à avaliação e certificação se atingirem soluções esperadas, previamente estabelecidas.

Mas, ainda não é o bastante. Testes durante o período de implantação poderão mostrar eventuais falhas, principalmente aquelas relacionadas com os aspectos específicos dos equipamentos envolvidos e da comunicação com os usuários. E durante o seu funcionamento em regime, testes programados deverão ser efetuados para se garantir a qualidade e a confiabilidade do sistema.

Sugestões para uma boa prática de programação:

- usar a modularização para tratar problemas complexos;
- usar estruturas básicas de controle em cada trecho de código;
- usar definições locais em módulos (evitar definições globais);
- usar a passagens de parâmetros para dentro e fora dos módulos;
- preferir a passagem de parâmetros por valor ao invés da passagem por referência;
- dar nomes às constantes;
- buscar clareza e simplicidade;
- identificar pré-condições e pós-condições, e verificá-las;
- documentar tudo durante o processo:
  - autoria, objetivos, datas, especificações, testes, referências etc.;
  - entrada e saída de cada módulo, e suas condições de uso;
  - explicar trechos importantes, ou detalhes não-óbvios;
- apresentar o código de maneira estética e legível;
  - usar um só comando por linha, quando possível;
  - usar maiúsculas e minúsculas para melhorar a legibilidade;
  - escolher bem os nomes de identificadores;
  - delimitar blocos em linhas separadas e com endentação;
  - fazer definições no início de cada bloco, quando possível;
  - fazer definições uma em cada linha, quando possível;
  - fazer bom uso das definições de tipos e classes;
  - separar os operadores em uma expressão, quando possível;
  - identificar cada entrada e saída de um programa;
- testar cada módulo separadamente e após integrá-los;
- testar casos especiais e documentá-los devidamente;
- avaliar a complexidade e estimar seu custo de execução.

Exemplo 1.

Fazer um algoritmo para:
- ler vários valores inteiros do teclado, um de cada vez;
- o último valor será igual a zero;
- calcular e mostrar a soma de todos os valores válidos.

Análise de dados:

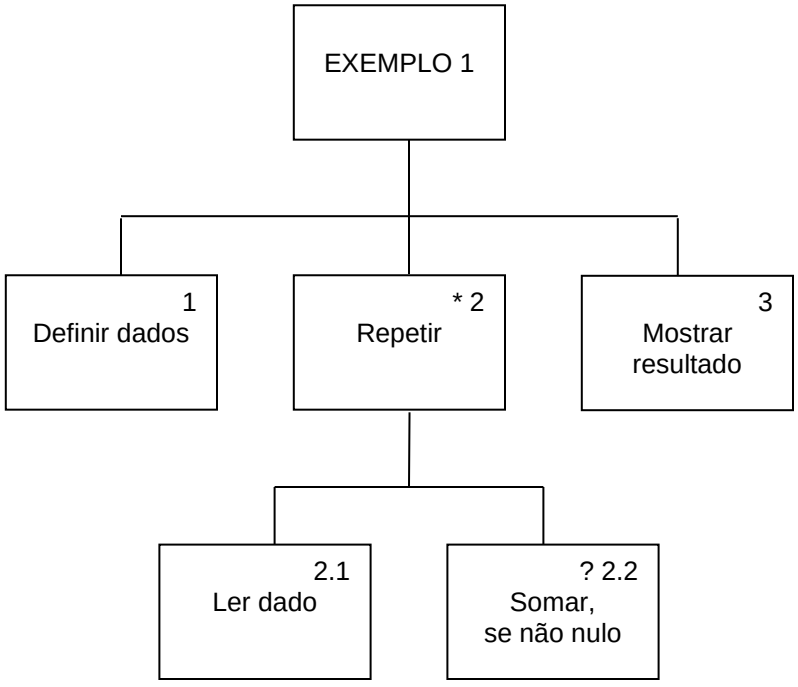
- Dados do problema :

Dado	Tipo	Valor Inicial	Obs.
X	inteiro		valor lido
SOMA	inteiro	0	soma dos valores lidos

- Fórmulas que relacionam os dados :

SOMA = SOMA + X; para cada valor lido de X diferente de zero

Diagrama funcional:



- Avaliação da solução :

  - Para teste podem ser usados os seguintes valores:

Dados	Resultado
1	
2	
3	
0	
	SOMA = 6

Algoritmo:

Esboço:

Primeira versão, só comentários.

Exemplo 1	v.1
Ação	Bloco
! definir dados	1
! repetir até ser nulo	2
! ler dados do teclado	2.1
! somar se for diferente de zero	2.2
! mostrar resultado	3

Segunda versão, refinar o primeiro bloco.

Exemplo 1	v.2
Ação	Bloco
! definir dados inteiro X, ! valor lido SOMA = 0; ! soma dos valores lidos	1
! repetir até ser nulo	2
! ler dados do teclado	2.1
! somar se for diferente de zero	2.2
! mostrar resultado	3

Terceira versão, refinar o segundo bloco.

Exemplo 1			v.3
Ação			Bloco
! definir dados inteiro X, ! valor lido SOMA = 0; ! soma dos valores lidos			1
! repetir até ser nulo			2
! ler dados do teclado tela ← "\nX="; X ← teclado ; ! ler valor			2.1
! somar se for diferente de zero			2.2
X ≠ 0 ?	V	SOMA = SOMA + X;	
X ≠ 0 ?			
! mostrar resultado			3

Quarta versão, refinar novamente o segundo bloco.

Exemplo 1	v.4
Ação	Bloco

! definir dados
  inteiro X, ! valor lido
  SOMA = 0; ! soma dos valores lidos
	1

! repetir enquanto não for nulo	2
! ler dados do teclado tela ← "\nX="; X ← teclado; ! ler valor	2.1
! somar se for diferente de zero se (X ≠ 0) ! bloco alternativo SOMA = SOMA + X; fim se ! X ≠ 0	2.2
X ≠ 0 ?	
! mostrar resultado	3

Quinta versão, refinar mais uma vez segundo bloco.

Exemplo 1	v.5
Ação	Bloco

! definir dados
  inteiro X, ! valor lido
  SOMA = 0; ! soma dos valores lidos
	1

! repetir enquanto não for nulo	2

repetir até X = 0
! bloco repetitivo
! 2.1. ler dados do teclado
  tela ← "\nX="; X ← teclado ; ! ler valor
! 2.2. somar se for diferente de zero
  se (X ≠ 0)
  ! bloco alternativo
  SOMA = SOMA + X;
  fim se ! X ≠ 0
fim repetir ! até X = 0	

! mostrar resultado tela ← ("\nSoma = “, SOMA);	3

Programa em SCILAB:

```
// Exemplo 1
// Calcular a soma de valores inteiros diferentes de zero.
//
//
// 1. definir dados
   X = 0;                        // valor lido
   SOMA = 0;                     // soma dos valores lidos
// 2.repetir ate' ser nulo
   // 2.1. ler dado do teclado
      X = input ( “\nX=“ );      // ler primeiro valor
      while ( X ~= 0 )           // repetir ate' X != 0
      // bloco repetitivo
      // 2.2. somar se for diferente de zero
         if ( X ~= 0 )
         // bloco alternativo
            SOMA = SOMA + X;
         end // fim do bloco alternativo
      // ler dado do teclado
         X = input ( “\nX=“ );    // ler outro valor
   end // fim do bloco repetitivo
// 3. mostrar resultado
   printf ( “\nSoma = %f”, SOMA );
// pausa para terminar
   printf ( “\nPressionar ENTER para terminar.\n” );
   halt;
// fim do programa
```

Programa em C:

```
// Exemplo 1
// Calcular a soma de valores inteiros diferentes de zero.
//
// bibliotecas necessárias
#include <stdio.h>
#include <stdio.h>
//
int main (void)
{
// 1. definir dados
   int X,              // valor lido
       SOMA = 0; // soma dos valores lidos
// 2.repetir até ser nulo
   do
   { // bloco repetitivo
     // ler dado do teclado
     printf ( "\nX=" );   scanf ( "%d", &X ); // ler valor
     // somar se for diferente de zero
     if (X != 0)
     { // bloco alternativo
       SOMA = SOMA + X;
     } // fim do bloco alternativo
   } // fim do bloco repetitivo
   while (X != 0);
// 3. mostrar resultado
   printf ( “\nSoma = %d“, SOMA );
// pausa para terminar
   printf ( "Pressionar ENTER para terminar." );
   getchar ( );
   return ( 0 );
} // fim do programa
```

Programa em C++:

```
// Exemplo 1
// Calcular a soma de valores inteiros diferentes de zero.
//
// bibliotecas necessárias
#include <iostream>
using namespace std;
//
int main (void)
{
// 1. definir dados
   int X,              // valor lido
       SOMA = 0; // soma dos valores lidos
// 2.repetir ate' ser nulo
   do
   { // bloco repetitivo
     // ler dado do teclado
     cout << "\nX=";   cin >> X; // ler valor
     // somar se for diferente de zero
     if (X != 0)
     { // bloco alternativo
       SOMA = SOMA + X;
     } // fim do bloco alternativo
   } // fim do bloco repetitivo
   while (X != 0);
// 3. mostrar resultado
   cout << “\nSoma = “ << SOMA;
// pausa para terminar
   cout << "Pressionar ENTER para terminar.";
   cin.get ( );
   return EXIT_SUCCESS;
} // fim do programa
```

Programa em C#:

```
/*
 * Exemplo 1
 * Dados dois resistores, calcular o resistor equivalente em serie.
 */

using System;

class Exemplo_1
{
   public static void Main ( )
   {
   // 1. definir dados
      int X,              // valor lido
          SOMA = 0; // soma dos valores lidos
   // 2. repetir ate' ser nulo
      do
      { // bloco repetitivo
      // 2.1. ler dados do teclado
         X = int.Parse ( Console.ReadLine ( ); // ler valor
      // 2.2. somar se for diferente de zero
         if ( X != 0 )
         { // bloco alternativo
            SOMA = SOMA + X;
         } // fim do bloco alternativo
      } // fim do bloco repetitivo
      while ( X != 0 );
   // 3. mostrar resultado
      Console.WriteLine ( "\nSOMA=" + SOMA );
   // pausa para terminar
      Console.Write ( "\nPressionar ENTER para terminar." );
      Console.ReadLine ( );
   } // end Main ( )

} // fim Exemplo_1 class
```

Programa em Java:

```
/**
  * Exemplo 1
  * Calcular a soma de valores inteiros diferentes de zero.
  */

// ---------------------------------------------- classes necessarias

// ---------------------------------------------- definicao de classe

class Exemplo_1
{
   public static void main ( String [ ] args )
   {
   // 1. definir dados
      int X,                    // valor lido
          SOMA = 0;        // soma dos valores lidos
   // 2. repetir ate' ser nulo
      do
      { // bloco repetitivo
      // 2.1. ler dados do teclado
         System.out.print ( "\nX = " );                    // ler valor
         X = Integer.parseInt ( System.console( ).readLine( ) );
      // 2.2. somar se for diferente de zero
         if ( X != 0 )
         { // bloco alternativo
            SOMA = SOMA + X;
         } // fim do bloco alternativo
      } // fim do bloco repetitivo
      while ( X != 0 );
   // 3. mostrar resultado
      System.out.println ( "\nSOMA = " + SOMA );
   // pausa para terminar
      System.out.print ( "\nPressionar ENTER para terminar." );
      System.console( ).readLine( );
   } // end main ( )

} // fim Exemplo_1 class
```

Programa em Python

```
# Exemplo 1
# Calcular a soma de valores inteiros diferentes de zero.
#
#
# 1. definir dados
X       = 1;       # valor a ser lido
                   # OBS.: O valor inicial servira' para
                   #           o controle da repeticao
SOMA = 0;       # soma dos valores lidos
# 2.repetir ate' ser nulo
while ( X != 0 ):                       # repetir ate' X == 0
  # bloco repetitivo
  # 2.1. ler dado do teclado
     X = int ( input ( "\nX=" ) );      # ler valor
  # 2.2. somar se for diferente de zero
     if ( X != 0 ):
     # bloco alternativo
        SOMA = SOMA + X;
     # fim do bloco alternativo
  # fim do bloco repetitivo
# 3. mostrar resultado
print  ( "\nSoma = ", SOMA );
# pausa para terminar
print  ( "\nPressionar ENTER para terminar.\n" );
input ( );
# fim do programa
```

Exercícios

1. Fazer um algoritmo para :
   - ler um número indeterminado de dados, contendo cada um, um valor inteiro;
   - o último valor será igual a zero e não deverá ser considerado;
   - calcular e mostrar apenas a soma dos valores positivos.

2. Fazer um algoritmo para :
   - ler um número indeterminado de dados, contendo cada um, um valor inteiro;
   - o último valor será igual a zero e não deverá ser considerado;
   - calcular e mostrar apenas a soma dos valores negativos.

3. Fazer um algoritmo para :
   - ler um número indeterminado de dados, contendo cada um, um valor inteiro;
   - o último valor será igual a zero e não deverá ser considerado;
   - calcular e mostrar, separadamente, a soma dos valores positivos e negativos.

4. Fazer um algoritmo para :
   - ler um número indeterminado de dados, contendo cada um, um valor inteiro;
   - o último valor será igual a zero e não deverá ser considerado;
   - calcular e mostrar apenas a soma dos valores pares.

5. Fazer um algoritmo para :
   - ler um número indeterminado de dados, contendo cada um, um valor inteiro;
   - o último valor será igual a zero e não deverá ser considerado;
   - calcular e mostrar apenas a soma dos valores ímpares.

Macros, expansões e substituições

Macros são formas de definição que permitem substituir uma porção de texto por uma palavra, ou combinação de palavras. No momento em que o conteúdo por ela(s) expresso for necessário, a definição é expandida naquele local. Normalmente, todo o texto de um algoritmo é expandido antes de ser executado, por isso, as macros devem ser as primeiras definições a aparecer em seu corpo.

Opcionalmente, uma macro pode ser expandida com a substituição de determinadas palavras que lhe serão passadas como parâmetros.

Observações :

- Todo o texto contido pela definição da macro será expandido, inclusive comentários;
- A expansão e substituição ocorrem ***apenas*** sobre o texto associado à macro.

Exemplo:

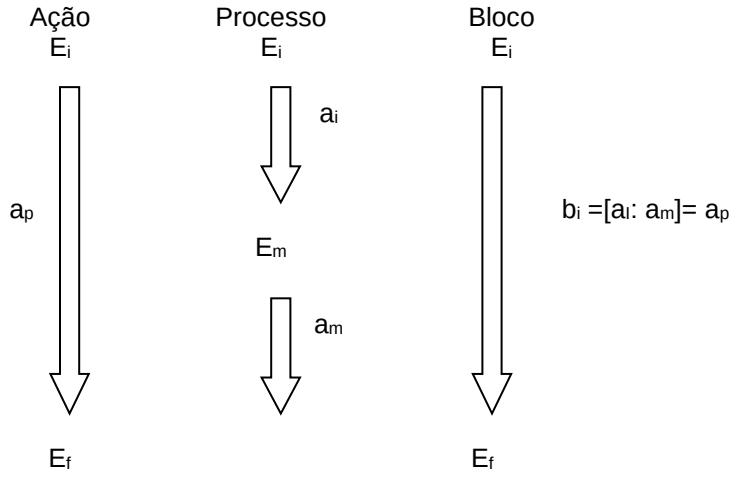
Programa em C++:

```
// Calcular a soma de valores inteiros diferentes de zero.
//
// bibliotecas necessárias
#include <iostream>
using namespace std;
//
// definições de macros
#define   NULO              0          // sem parâmetro
#define   NÃO_NULO(X)    (X != 0) // com parâmetro
//
int main (void)
{
// 1. definir dados
   int X,                                  // valor lido
       SOMA = 0;                           // soma dos valores lidos
// 2.repetir ate' ser nulo
   do
   { // bloco repetitivo
     // ler dados do teclado
     cout << "\nX=";   cin >> X;          // ler valor
     // somar se for diferente de zero
     if (X != NULO)                        // o mesmo que ( X != 0 )
     { // bloco alternativo
       SOMA = SOMA + X;
     } // fim do bloco alternativo
   } // fim do bloco repetitivo
   while ( NÃO_NULO(X) );              // o mesmo que ( X != 0 )
// 3. mostrar resultado
   cout << “\nSoma = “ << SOMA;
// pausa para terminar
   cout << "Pressionar ENTER para terminar.";
   getchar ( );
   return EXIT_SUCCESS;
} // fim do programa
```

Modularização

Os conceitos de ação primitiva e de processo seqüencial podem ser associados ao conceito de abstração de comandos.

Se, antes, na inexistência de uma determinada ação primitiva, poderíamos substituí-la por um processo seqüencial equivalente; agora, podemos agrupar as (sub)ações deste processo, de modo a criar um bloco equivalente àquela ação desejada.



Esta possibilidade de decompor um ação em outras menores, e poder reagrupá-las em unidades sintaticamente fechadas e logicamente relacionadas, chama-se modularização, e pode trazer uma série de vantagens a um algoritmo :

- aumentar a legibilidade, a eficiência e portabilidade;
- facilitar a execução de testes e sua manutenção;
- permitir o isolamento de erros e a sua correção;
- permitir compartilhamento e desenvolvimento concorrentes.

A modularização pode tornar uma solução bem mais fácil de ser alcançada, supondo que as partes menores possuam uma complexidade menor que a do problemas original. Quanto mais independentes estas partes (os módulos), mais fácil é o desenvolvimento de cada uma delas, podendo, inclusive, ficar a cargo de pessoas diferentes para fazê-lo.

Ao grau de interdependência das partes chama-se ***acoplamento***, e quanto menor, mais fácil a manutenção. Serve como fator de medida de qualidade. Alguns fatores podem influenciar neste grau:

- quantidade de dados e comandos;
- complexidade das partes;
- tipo de acoplamento : por dados, por controle etc.

Ao grau de relacionamento entre as entidades e instruções agrupadas em um bloco chama-se ***coesão***, e quanto maior, melhor a estrutura. Este também é um fator de qualidade. Há vários tipos de coesão :

- coincidência : não existe coesão;
- lógica : em um classe de operações;
- temporal : relacionadas pela execução;
- procedimental : execução seqüencial de comandos;
- comunicação : operações sobre os mesmos dados;
- seqüencial : a saída de um bloco é a entrada de outro;
- funcional : para os mesmos dados, mesmo objetivo.

Função

Freqüentemente, na Matemática, utiliza-se o conceito de função como sendo uma relação definida entre dois domínios distintos. O sentido empregado aqui é uma extensão do conceito.

Exemplo:

Supor a tabela abaixo :

x	y
0	0
1	1
2	4
3	9
4	16
5	25

Se a variável (y) for função de (x), pode-se dizer que :

$$y = f(x) = x^2$$

ou expressa na sua forma algorítmica na forma de um operador infixo (entre os operandos):

Y = X ^ 2

como visto anteriormente em expressões aritméticas. Entretanto, esta expressão é muito particular, não servindo a quaisquer variáveis, mas apenas aos valores contidos nestas duas (X, Y). Assim, se fosse necessário representar a mesma relação entre duas outras variáveis, por exemplo, A e B, seria preciso reescrever a expressão :

A = B ^ 2

Se houvesse um meio de definir apenas a relação, independente das variáveis envolvidas, poderíamos reutilizá-la, quantas vezes fosse necessário, substituindo, apenas no momento de execução, os valores desejados, não importando qual nome de variável tenha sido usado quando da definição.

A definição de uma função também deve preceder qualquer outro comando executável.

Sua ação é executada apenas quando for feita uma chamada, ou seja, o uso do nome da função em uma expressão. Uma função sempre retorna um valor de resultado no próprio nome, no lugar onde foi chamada.

Durante a chamada, primeiro ocorre uma expansão, com substituição de parâmetros, se houver, na memória; depois, executa-se o código, calculando-se o resultado, que é retornado ao ponto onde foi feita a chamada, e libera-se a memória ocupada durante esta execução.

Uma função somente pode ser chamada para substituir a determinação de um valor.

Exemplo:

```
real função QUADRADO ( real X )
      retornar ( X * X );
fim função ! QUADRADO
```

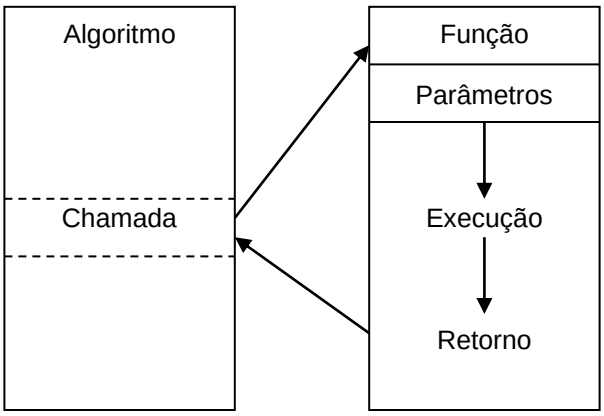
Função em C/C++:

```
float QUADRADO ( float X )
{
      return ( X * X );
} // fim QUADRADO ( )
```

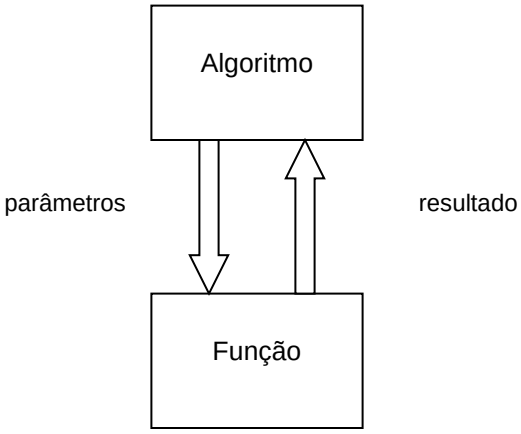
Função em SCILAB:

```
function retorno = QUADRADO ( X )
      retorno = X * X ;
// fim QUADRADO ( )
```

- Funcionamento :



- Diagrama de relacionamento :



- Funções sem parâmetros

A forma mais simples de utilização deste recurso é supor o encapsulamento de comandos, com definição de um determinado valor, equivalente a uma atribuição.

Exemplo:

```
real função PI ( )
      retornar ( 3.141592 );
fim função ! PI
```

No trecho acima foi utilizada uma constante (3.141592), mas também poderia ser utilizada uma variável global, uma expressão ou a chamada de outra função que determinasse o mesmo valor.

Se, ao invés da constante, fosse desejado um valor mais preciso poderia ser usada uma função predefinida – atan(x), que calcula o arco-tangente, por exemplo :

Exemplo:

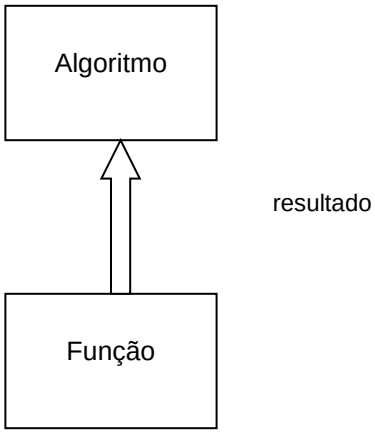
```
real função PI ( )
      retornar ( 4 * atan(1.0) );
fim função ! PI ( )
```

Utilizando o conceito de função, uma atribuição de valor poderia ser feita de uma forma mais flexível, apenas encapsulando comandos.

Exemplo:

```
inteiro função Ler_Positivo ( )
   inteiro X;                ! variável auxiliar local
   repetir até X > 0
      X ← teclado;
   fim repetir               ! repetir a leitura enquanto não for positivo
  retornar ( X );
fim função ! fim Ler_Positivo ( )
```

Reparar que, independente de como o valor for estabelecido, vale o diagrama de relacionamento abaixo :



Exemplos em SCILAB:

```
function retorno = PI
      retorno = 3.141592;
endfunction // PI ( )

function retorno = PI4
      retorno =  4 * atan(1.0);
endfunction // P4I ( )

function retorno = LerPositivo
//
  X = input ( “\nFornecer um valor positivo: “ );
  while ( X <= 0 )            // repetir ate' (X > 0)
      X = input ( “\nFornecer um valor positivo: “ );
  end                         // fim repetir enquanto nao for positivo
  retorno = X ;
endfunction // Ler_Positivo ( )
```

Exemplos em C:

```
#include <math.h>

float PI ( )
{
      return ( 3.141592 );
} // fim PI ( )

float PI4 ( )
{
      return (4 * atan(1.0));
} // fim PI4 ( )

int Ler_Positivo ( )
{
   int X;                     // auxiliar local
   do                         // repetir ate' (X > 0)
     cin >> X;
   while ( X <= 0 );          // fim repetir enquanto não for positivo
  return ( X );
} // fim Ler_Positivo ( )
```

Exemplos em C++: (usar math.h)

```
float PI ( )
{
      return ( 3.141592 );
} // fim PI ( )

float PI4 ( )
{
      return (4 * atan(1.0));
} // fim PI4 ( )

int Ler_Positivo ( )
{
   int X;                    // auxiliar local
   do                        // repetir ate' (X > 0)
     cin >> X;
   while ( X <= 0 );         // fim repetir enquanto não for positivo
  return ( X );
} // fim Ler_Positivo ( )
```

Exemplos em C#: (usar Math)

```
public static double PI ( )
{
      return ( 3.141592 );
} // fim PI ( )

public static double PI4 ( )
{
      return ( 4 * Math.Atan (1.0) );
} // fim PI4 ( )

public int Ler_Positivo ( )
{
   int X;                    // auxiliar local
   do                        // repetir ate' (X > 0)
                             // ler o valor  (using System)
     X = int.Parse ( Console.ReadLine ( ) );
   while ( X <= 0 );         // fim repetir enquanto não for positivo
  return ( X );
} // fim Ler_Positivo ( )
```

Exemplos em Java: (usar Math)

```
public static double PI ( )
{
      return ( 3.141592 );
} // fim PI ( )

public static double PI4 ( )
{
      return ( 4 * Math.atan (1.0) );
} // fim PI4 ( )

publicd int Ler_Positivo ( )
{
   int X;                        // auxiliar local
   do                            // repetir ate' (X > 0)
      X = Integer.parseInt ( System.console( ).readLine( ) );
   while ( X <= 0 );             // fim repetir enquanto não for positivo
  return ( X );
} // fim Ler_Positivo ( )
```

Exemplos em Python: (usar math)

```
from math import *

def PI ( ):
  return 3.141592;
# PI ( )

def PI4 ( ):
  return ( 4.0 * atan(1.0) );
# PI ( )

def LerPositivo ( ):
#
  X = int ( input ( "\nFornecer um valor positivo: " ) );
  while ( X <= 0 ):             # repetir ate' (X > 0)
    X = int ( input ( "\nFornecer um valor positivo: " ) );
  # fim repetir enquanto nao for positivo
  return X ;
# Ler_Positivo ( )
```

Exemplo 2.

Fazer um algoritmo para:
- implementar uma função para calcular a soma do 10 primeiros números inteiros;
- calcular e mostrar o valor calculado.

Análise de dados:

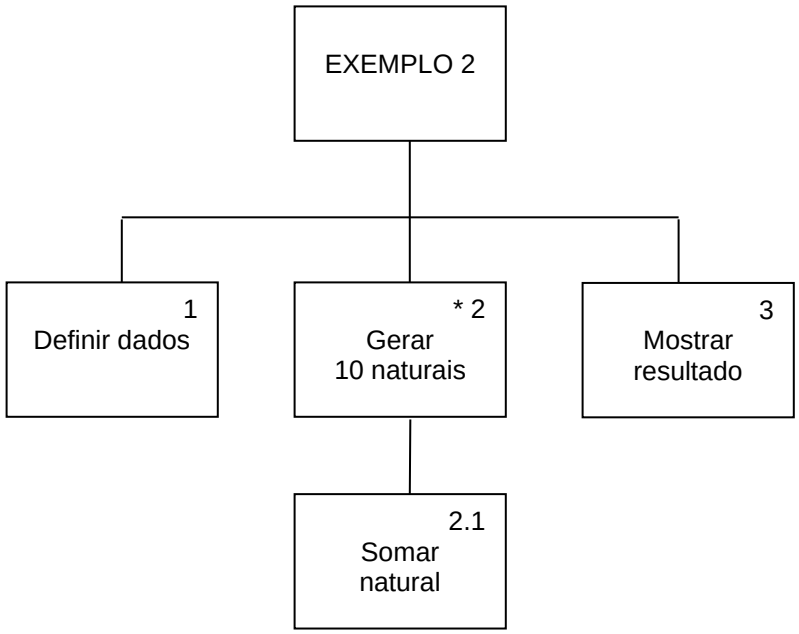
- Dados da função:

Dado	Tipo	Valor Inicial	Obs.
X	inteiro		contador
SOMA	inteiro		soma dos termos

- Fórmulas que relacionam os dados :

SOMA = SOMA + X; para cada número natural menor ou igual a 10

Diagrama funcional:



- Avaliação da solução :

SOMA = 1+2+3+4+5+6+7+8+9+10 = 55

Algoritmo, sem função, a princípio:

Esboço:

Primeira versão, só comentários.

Exemplo 2	v.1
Ação	Bloco
! definir dados	1
! gerar 10 números naturais	2
! somar cada número natural	2.1
! mostrar resultado	3

Segunda versão, refinar o primeiro bloco.

Exemplo 2	v.2
Ação	Bloco
! definir dados inteiro X, ! número natural SOMA = 0; ! soma dos naturais	1
! gerar 10 números natural	2
! somar cada número natural	2.1
! mostrar resultado	3

Terceira versão, refinar o segundo bloco.

Exemplo 2	v.3
Ação	Bloco
! definir dados inteiro X, ! valor lido SOMA = 0; ! soma dos valores lidos	1
! gerar 10 números naturais	2
X = 1 : 10 : 1	
! somar cada número natural SOMA = SOMA + X;	2.1
! mostrar resultado	3

Quarta versão, refinar novamente o segundo bloco.

Exemplo 2	v.4
Ação	Bloco
! definir dados inteiro X, ! valor lido SOMA = 0; ! soma dos valores lidos	1
! gerar 10 números naturais	2
repetir para (X = 1 : 10 : 1) ! somar cada número natural SOMA = SOMA + X; fim repetir ! para (X = 1 : 10 : 1)	2.1
! mostrar resultado	3

Quinta versão, refinar o terceiro bloco.

Exemplo 2	v.5
Ação	Bloco

! definir dados
  inteiro X, ! valor lido
  SOMA = 0; ! soma dos valores lidos
	1

! gerar 10 números naturais	2
repetir para (X = 1 : 10 : 1) ! somar cada número natural SOMA = SOMA + X; fim repetir ! para (X = 1 : 10 : 1)	2.1

! chamada da função para mostrar resultado
  tela ← (“\nSoma dos 10 primeiros naturais =”.
  SOMAR10( );
	3

Transformação em função.

Exemplo 5	v.6
Ação	Bloco
inteiro função SOMAR10 ( ) ! sem parâmetros	

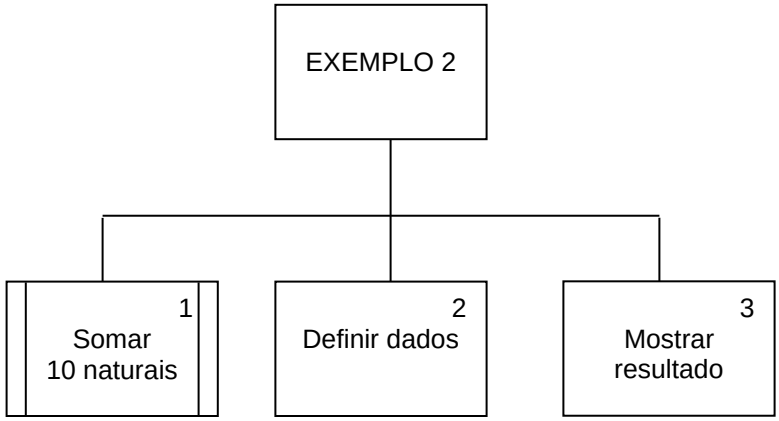
! definir dados locais
  inteiro X, ! valor lido
  SOMA = 0; ! soma dos valores lidos
	1

! gerar 10 números naturais	2
repetir para (X = 1 : 10 : 1) ! somar cada número natural SOMA = SOMA + X; fim repetir ! para (X = 1 : 10 : 1)	
retornar (SOMA);	
! parte principal	

! chamada da função para mostrar resultado
  tela ← (“\nSoma dos 10 primeiros naturais =”.
  SOMAR10( );
	3

A função tornou o programa mais complexo, no entanto, as variáveis locais só existem enquanto estiver ativa, após uma chamada. Ela pode ser chamada várias vezes e o comportamento do algoritmo interno será o mesmo.

Novo diagrama funcional:



Programa em SCILAB:

```
// Exemplo 2.
// Calcular a soma dos 10 primeiros numeros naturais.
//
// definir funcao sem parametro
function retorno = SOMAR10 ( )
// 1. definir dados locais
   X = 0;              // numero natural
   SOMA = 0;       // soma dos valores lidos
// 2.gerar 10 numeros naturais
   for ( X = 1 : 1 : 10 ),
   // 2.1 somar cada numero natural
      SOMA = SOMA + X;
   end // fim do bloco repetitivo
   // retorno do valor calculado
   retorno = SOMA;
endfunction // SOMAR10 ( )
//
// parte principal
//
// 3. mostrar resultado
    clc;                // limpar a area de trabalho
    // chamada da funcao, sem parametro
    printf ( "\nSoma dos 10 primeiros naturais = %f", SOMAR10 ( ) );
// pausa para terminar
    printf ( "\nPressionar ENTER para terminar.\n" );
    halt;
// fim do programa
```

Programa em C:

```
// Exemplo 2.
// Calcular a soma dos 10 primeiros numeros naturais.
//
// bibliotecas necessarias
#include <stdio.h>
#include <stdlib.h>
//
// definir funcao sem parametro
int SOMAR10 (void)
{
 // 1. definir dados locais
   int X,                          // numero natural
       SOMA = 0;                   // soma dos valores lidos
 // 2.gerar 10 numeros naturais
   for ( X = 1; X <= 10; X = X+1 )
   {
     // 2.1 somar cada numero natural
     SOMA = SOMA + X;
   } // fim do bloco repetitivo
   // retorno do valor calculado
   return ( SOMA );
} // fim SOMAR10( )
//
// parte principal
//
int main (void)
{
// 3. mostrar resultado
   printf  ( “\nSoma dos 10 primeiros naturais = %d“,
               SOMAR10( ) ); // chamada da funcao, sem parametro
// pausa para terminar
   printf ( "Pressionar ENTER para terminar." );
   getchar ( );
   return ( 0 );
} // fim do programa
```

Programa em C++:

```
// Exemplo 2.
// Calcular a soma dos 10 primeiros numeros naturais.
//
// bibliotecas necessarias
#include <iostream>
using namespace std;
//
// definir funcao sem parametro
int SOMAR10 (void)
{
 // 1. definir dados locais
   int X,                          // numero natural
       SOMA = 0;                   // soma dos valores lidos
 // 2.gerar 10 numeros naturais
   for ( X = 1; X <= 10; X = X+1 )
   {
     // 2.1 somar cada numero natural
     SOMA = SOMA + X;
   } // fim do bloco repetitivo
   // retorno do valor calculado
   return ( SOMA );
} // fim SOMAR10( )
//
// parte principal
//
int main (void)
{
// 3. mostrar resultado
   cout << "\nSoma dos 10 primeiros naturais ="
        << SOMAR10( );     // chamada da funcao, sem parametro
// pausa para terminar
   cout << "Pressionar ENTER para terminar.";
   getchar ( );
   return EXIT_SUCCESS;
} // fim do programa
```

Programa em C#:

```
/*
 * Exemplo 2
 * Calcular a soma dos 10 primeiros numeros naturais.
 * Dados valores de resistores, calcular o valor medio.
 */
using System;

class Exemplo_2
{
// definir funcao sem parametro
   static int SOMAR10 ( )
   {
   // 1. definir dados locais
      int X,             // numero natural
          SOMA = 0; // soma dos valores lidos
   // 2. gerar 10 numeros naturais
      for ( X = 1; X <= 10; X = X+1 )
      {
      // 2.1. somar cada numero natural
         SOMA = SOMA + X;
      } // fim do bloco repetitivo
   // retorno do valor calculado
      return ( SOMA );
   } // fim SOMAR10 ( )
//
// parte principal
//
   public static void Main ( )
   {
   // 3. mostrar resultado
      Console.WriteLine ( "\nSoma dos 10 primeiros naturais = " +
                                 SOMAR10( ) ); // chamada de funcao sem parametro
   // pausa para terminar
      Console.Write ( "\nPressionar ENTER para terminar." );
      Console.ReadLine ( );
   } // end Main ( )

} // fim Exemplo_2 class
```

Programa em Java:

```
/**
  * Exemplo 2
  * Calcular a soma dos 10 primeiros numeros naturais.
  */

// ------------------------------------------------ classes necessarias

// ------------------------------------------------ definicao de classe

class Exemplo_2
{
// definir funcao sem parametro
   public static int SOMAR10 ( )
   {
   // 1. definir dados locais
      int X,               // numero natural
          SOMA = 0;    // soma dos valores lidos
   // 2. gerar 10 numeros naturais
      for ( X = 1; X <= 10; X = X+1 )
      {
      // 2.1. somar cada numero natural
         SOMA = SOMA + X;
      } // fim do bloco repetitivo
   // retorno do valor calculado
      return ( SOMA );
   } // fim SOMAR10 ( )
//
// parte principal
//
   public static void main ( String [ ] args )
   {
   // 3. mostrar resultado
      System.out.println ( "\nSoma dos 10 primeiros naturais = " +
                           SOMAR10( ) ); // chamada de funcao sem parametro
   // pausa para terminar
      System.out.print ( "\nPressionar ENTER para terminar." );
      System.console( ).readLine( );
   } // end main ( )

} // fim Exemplo_2 class
```

Programa em Python:

```
# Exemplo 2.
# Calcular a soma dos 10 primeiros numeros naturais.
#
# definir funcao sem parametro
def SOMAR10 ( ):
# 1. definir dados locais
   X    = 0;          # numero natural
   SOMA = 0;     # soma dos valores lidos
# 2.gerar 10 numeros naturais
   for X in range ( 1, 10+1, 1 ):
   # 2.1 somar cada numero natural
      SOMA = SOMA + X;
   # fim do bloco repetitivo
# retorno do valor calculado
   return ( SOMA );
# SOMAR10 ( )
#
# parte principal
#
# 3. mostrar resultado
#     chamada da funcao, sem parametro
print ( "\nSoma dos 10 primeiros naturais = ", SOMAR10 ( ) );
# pausa para terminar
print  ( "\nPressionar ENTER para terminar.\n" );
input ( );
# fim do programa
```

- Funções com parâmetros

A utilização de parâmetros permitirá uma flexibilidade ainda maior no uso de funções.

O número e tipo dos parâmetros podem variar.

Exemplo:

```
real função hipotenusa_do_triangulo_retangulo_equilatero (inteiro lado)
      retornar ( raiz (2 * lado*lado) );
fim função ! hipotenusa_do_triangulo_retangulo_equilatero
```

Exemplo:

```
real função hipotenusa_de_triangulo_retangulo (real a, real b)
      retornar ( raiz( a*a + b*b ) );
fim função ! hipotenusa_de_triangulo_retangulo
```

Exercícios:

1. Escrever uma função para calcular a hipotenusa de um triângulo retângulo dado um lado e sabendo-se que o outro lado é o dobro deste.

2. Escrever uma função para calcular a hipotenusa de um triângulo retângulo dados um lado e o ângulo entre ele a hipotenusa.

3. Escrever uma função para calcular o ângulo entre a hipotenusa de um triângulo retângulo e um dos lados.

4. Escrever uma função para calcular um lado de um triângulo retângulo dados a hipotenusa e o seno adjacente entre eles.

5. Escrever uma função para calcular a área de um triângulo retângulo dados a hipotenusa e a tangente com um dos lados.

Exemplo 3.

Fazer um algoritmo para:
- implementar uma função para calcular a soma do 10 primeiros números inteiros;
- calcular e mostrar o valor calculado.

Análise de dados:

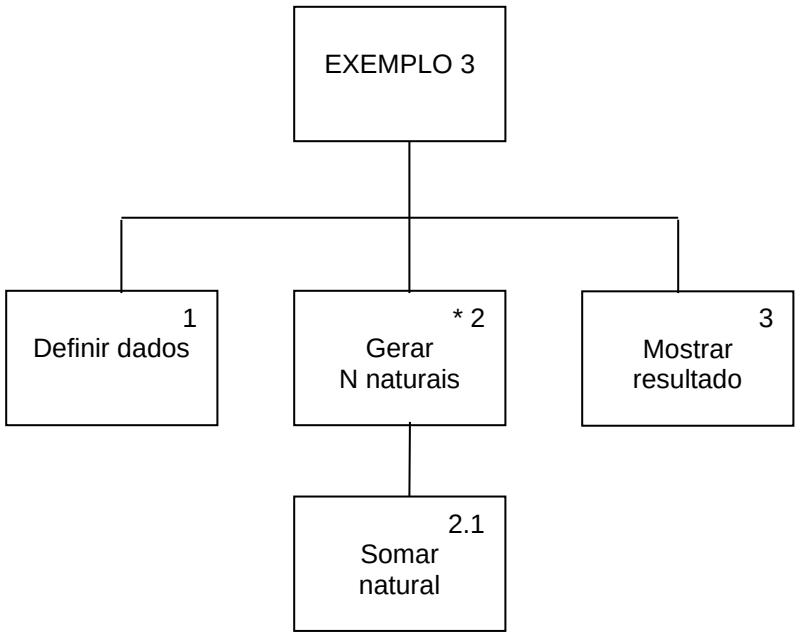
- Dados da função:

Dado	Tipo	Valor Inicial	Obs.
X	inteiro		contador
N	inteiro		número de termos
SOMA	inteiro		soma dos termos

- Fórmulas que relacionam os dados :

SOMA = SOMA + X; para cada número natural menor ou igual a 10

Diagrama funcional:



- Avaliação da solução :

SOMA = 1+2+3+4+5+6+7+8+9+10 = 55

Algoritmo, com função:

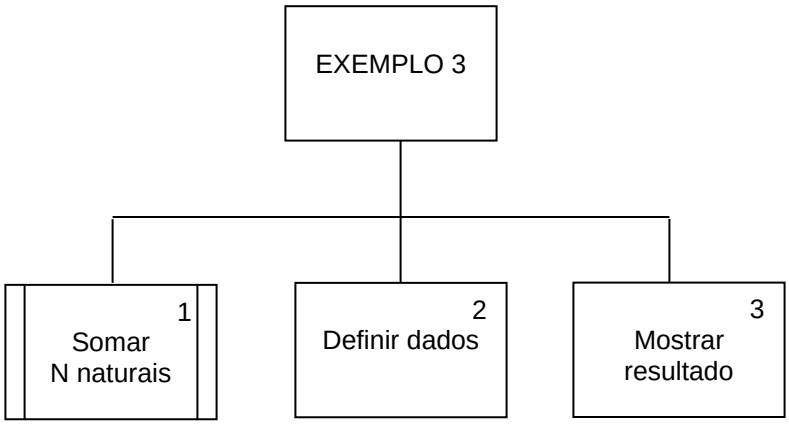
Esboço:

Modificação da função.

Exemplo 3	v.7
Ação	Bloco
inteiro função SOMAR (inteiro N) ! com parâmetros	
! definir dados locais inteiro X, ! valor lido SOMA = 0; ! soma dos valores lidos	1
! gerar N números naturais	2
repetir para (X = 1 : N : 1) ! somar cada número natural SOMA = SOMA + X; fim repetir ! para (X = 1 : N : 1)	
retornar (SOMA);	
! parte principal	
! chamada da função para mostrar resultado tela ← (“\nSoma dos 10 primeiros naturais =”. SOMAR (10);	3

A função ganhou versatilidade, pois não se limita o número de valores a serem somados.

Novo diagrama funcional:



Programa em SCILAB:

```
// Exemplo 3.
// Calcular a soma dos 10 primeiros numeros naturais.
//
// definir funcao com parametro
function retorno = SOMAR ( N )
// parametro:
//  int N;                      // numero de termos
//
// 1. definir dados locais
   X      = 0;                  // numero natural
   SOMA = 0;                    // soma dos valores lidos
// 2.gerar 10 numeros naturais
   for ( X = 1 : 1 : N ),
   // 2.1 somar cada numero natural
      SOMA = SOMA + X;
   end // fim do bloco repetitivo
   // retorno do valor calculado
   retorno = SOMA;
endfunction // SOMAR( )
//
// parte principal
//
// 3. mostrar resultado
   clc;                         // limpar a area de trabalho
   // chamada da funcao, com parametro
   printf ( "\nSoma dos 10 primeiros naturais = %f", SOMAR( 10 ) );
// pausa para terminar
   printf ( "\nPressionar ENTER para terminar.\n" );
   halt;
// fim do programa
```

Programa em C:

```
// Exemplo 3.
// Calcular a soma dos 10 primeiros numeros naturais.
//
// bibliotecas necessarias
#include <stdio.h>
#include <stdlib.h>
//
// definir funcao com parametro
int SOMAR ( int N )
{
 // 1. definir dados locais
   int X,                          // numero natural
       SOMA = 0;                   // soma dos valores lidos
 // 2.gerar N numeros naturais
   for ( X = 1; X <= N; X = X+1 )
   {
     // 2.1 somar cada numero natural
     SOMA = SOMA + X;
   } // fim do bloco repetitivo
   // retorno do valor calculado
   return ( SOMA );
} // fim SOMAR ( )
//
// parte principal
//
int main (void)
{
// calcular a soma dos 10 primeiros naturais
// 3. mostrar resultado
   printf  ( “\nSoma dos 10 primeiros naturais = %d“,
                SOMAR ( 10 ) );          // chamada da funcao, com parametro
// pausa para terminar
   printf ( "Pressionar ENTER para terminar." );
   getchar ( );
   return ( 0 );
} // fim do programa
```

Programa em C++:

```
// Exemplo 3.
// Calcular a soma dos 10 primeiros numeros naturais.
//
// bibliotecas necessarias
#include <iostream>
using namespace std;
//
// definir funcao com parametro
int SOMAR ( int N )
{
 // 1. definir dados locais
   int X,                          // numero natural
       SOMA = 0;                   // soma dos valores lidos
 // 2.gerar N numeros naturais
   for ( X = 1; X <= N; X = X+1 )
   {
     // 2.1 somar cada numero natural
     SOMA = SOMA + X;
   } // fim do bloco repetitivo
   // retorno do valor calculado
   return ( SOMA );
} // fim SOMAR ( )
//
// parte principal
//
int main (void)
{
// calcular a soma dos 10 primeiros naturais
// 3. mostrar resultado
   cout << "\nSoma dos 10 primeiros naturais ="
        << SOMAR ( 10 );    // chamada da funcao, com parametro
// pausa para terminar
   cout << "Pressionar ENTER para terminar.";
   cin.get ( );
   return EXIT_SUCCESS;
} // fim do programa
```

Programa em C#:

```
/*
 * Exemplo 3
 * Calcular a soma dos 10 primeiros numeros naturais.
 */
using System;

class Exemplo_9
{
// definir funcao com parametro
   public static int SOMAR ( int N )
   {
   // 1. definir dados locais
      int X,              // numero natural
          SOMA = 0; // soma dos valores lidos
   // 2. gerar N numeros naturais
      for ( X = 1; X <= N; X = X+1 )
      {
      // 2.1. somar cada numero natural
         SOMA = SOMA + X;
      } // fim do bloco repetitivo
   // retorno do valor calculado
      return ( SOMA );
   } // fim SOMAR ( )
//
// parte principal
//
   public static void Main ( )
   {
   // 3. mostrar resultado
      Console.WriteLine ( "\nSoma dos 10 primeiros naturais = " +
                                 SOMAR( 10 ) ); // chamada de funcao com parametro
   // pausa para terminar
      Console.Write ( "\nPressionar ENTER para terminar." );
      Console.ReadLine ( );
   } // end Main ( )

} // fim Exemplo_3 class
```

Programa em Java:

```
/**
  * Exemplo 3
  * Calcular a soma dos 10 primeiros numeros naturais.
  */

// ------------------------------------------------ classes necessarias

// ------------------------------------------------ definicao de classe

class Exemplo_3
{
// definir funcao com parametro
   public static int SOMAR ( int N )
   {
   // 1. definir dados locais
      int X,                        // numero natural
          SOMA = 0;                 // soma dos valores lidos
   // 2. gerar N numeros naturais
      for ( X = 1; X <= N; X = X+1 )
      {
      // 2.1. somar cada numero natural
         SOMA = SOMA + X;
      } // fim do bloco repetitivo
   // retorno do valor calculado
      return ( SOMA );
   } // fim SOMAR ( )
//
// parte principal
//
   public static void main ( String [ ] args )
   {
   // 3. mostrar resultado
      System.out.println ( "\nSoma dos 10 primeiros naturais = " +
                           SOMAR( 10 ) ); // chamada de funcao com parametro
   // pausa para terminar
      System.out.print ( "\nPressionar ENTER para terminar." );
      System.console( ).readLine( );
   } // end main ( )

} // fim Exemplo_3 class
```

Programa emPython:

```
# Exemplo 3.
# Calcular a soma dos 10 primeiros numeros naturais.
#
# definir funcao com parametro
def SOMAR ( N ):
# parametro:
#  int N;                    # numero de termos
#
# 1. definir dados locais
   X    = 0;                 # numero natural
   SOMA = 0;                 # soma dos valores lidos
# 2.gerar 10 numeros naturais
  for X in range ( 1, N+1, 1 ):
  # 2.1 somar cada numero natural
     SOMA = SOMA + X;
  # fim do bloco repetitivo
# retorno do valor calculado
   return ( SOMA );
# SOMAR( )
#
# parte principal
#
# 3. mostrar resultado
#     chamada da funcao, com parametro
print  ( "\nSoma dos 10 primeiros naturais = ", SOMAR( 10 ) );
# pausa para terminar
print  ( "\nPressionar ENTER para terminar.\n" );
input ( );
# fim do programa
```

Exercícios

1. Fazer um algoritmo para :
   - definir uma função para determinar se um número inteiro é positivo;
   - ler um número indeterminado de dados, contendo cada um, um valor inteiro;
   - o último valor será igual a zero e não deverá ser considerado;
   - usando a função, mostrar que valores lidos são positivos.

2. Fazer um algoritmo para :
   - definir uma função para determinar se um número inteiro é negativo;
   - ler um número indeterminado de dados, contendo cada um, um valor inteiro;
   - o último valor será igual a zero e não deverá ser considerado;
   - usando a função, mostrar que valores lidos são negativos.

3. Fazer um algoritmo para :
   - definir uma função para determinar se um número inteiro é par;
   - ler um número indeterminado de dados, contendo cada um, um valor inteiro;
   - o último valor será igual a zero e não deverá ser considerado;
   - usando a função, mostrar que valores lidos são pares.

4. Fazer um algoritmo para :
   - definir uma função para determinar se um número inteiro é ímpar;
   - ler um número indeterminado de dados, contendo cada um, um valor inteiro;
   - o último valor será igual a zero e não deverá ser considerado;
   - usando a função, mostrar que valores lidos são ímpares.

5. Fazer um algoritmo para :
   - definir uma função para determinar se um número inteiro é divisível por outro;
   - ler um número indeterminado de pares de dados inteiros;
   - o último par de dados será igual a (zero, zero) e não deverá ser considerado;
   - usando a função, mostrar que pares lidos têm o primeiro valor divisível pelo segundo.

Exemplo 4.

Fazer um algoritmo para:
- implementar uma função para calcular a soma de números naturais no intervalo [1, 10];
- calcular e mostrar o valor calculado.

Análise de dados:

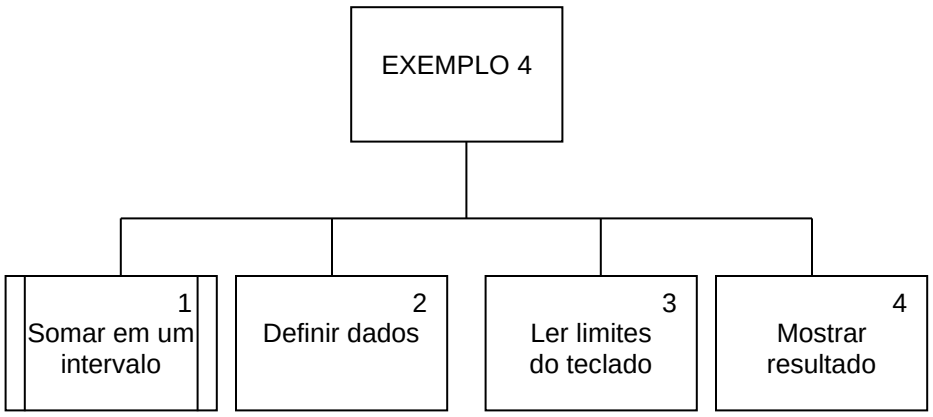
- Dados da função:

Dado	Tipo	Valor Inicial	Obs.
X	inteiro		contador
LI	inteiro		limite inferior
LS	inteiro		limite superior
SOMA	inteiro		soma dos termos

- Fórmulas que relacionam os dados :

SOMA = SOMA + X; para cada número natural no intervalo [LI, LS]

Diagrama funcional:



- Avaliação da solução :

SOMA = 1+2+3+4+5+6+7+8+9+10 = 55

Algoritmo, com função:

Esboço:

Modificação da função.

Exemplo 4	v.8
Ação	Bloco
inteiro SOMAR (inteiro LI, inteiro LS) ! função com dois parâmetros	
! definir dados locais inteiro X, ! valor lido SOMA = 0; ! soma dos valores lidos	1
! gerar N números naturais	2
repetir para (X = LI : LS : 1) ! somar cada número natural SOMA = SOMA + X; fim repetir ! para (X = LI : LS : 1)	
retornar (SOMA);	
! parte principal	
! chamada da função para mostrar resultado tela ← (“\nSoma dos 10 primeiros naturais =”. SOMAR (1, 10);	3

A função ganhou ainda maior versatilidade, pois não se limita aos valores a serem somados comecem sempre em 1.

Programa em SCILAB:

```
// Exemplo 4.
// Calcular a soma dos 10 primeiros numeros naturais.
//
// definir funcao com parametros
function retorno = SOMAR ( LI, LS )
// parametros:
//      int LI,             // limite inferior
//          LS;             // limite superior
//
// 1. definir dados locais
   X = 0;                   // numero natural
   SOMA = 0;                // soma dos valores lidos
// 2.gerar 10 numeros naturais
   for ( X = LI : 1 : LS ),
   // 2.1 somar cada numero natural
      SOMA = SOMA + X;
   end // fim do bloco repetitivo
   // retorno do valor calculado
   retorno = SOMA;
endfunction // SOMAR( )
//
// parte principal
//
// 3. mostrar resultado
   clc;                     // limpar a area de trabalho
   // chamada da função, com parametros
   printf ( "\nSoma dos 10 primeiros naturais = %f", SOMAR( 1, 10 ) );
// pausa para terminar
   printf ( "\nPressionar ENTER para terminar.\n" );
   halt;
// fim do programa
```

Programa em C:

```
// Exemplo 4.
// Calcular a soma dos 10 primeiros numeros naturais.
//
// bibliotecas necessárias
#include <iostream>
using namespace std;
//
// definir funcao com dois parametros
int SOMAR ( int LI, int LS )
{
 // 1. definir dados locais
   int X,                                  // numero natural
       SOMA = 0;                           // soma dos valores lidos
 // 2.gerar N numeros naturais
   for ( X = LI; X <= LS; X = X+1 )
   {
     // 2.1 somar cada numero natural
     SOMA = SOMA + X;
   } // fim do bloco repetitivo
   // retorno do valor calculado
   return ( SOMA );
} // fim SOMAR ( )
//
// parte principal
//
int main (void)
{
// calcular a soma dos 10 primeiros naturais
// 3. mostrar resultado
   printf  ( “\nSoma dos 10 primeiros naturais = %d“,
                SOMAR ( 1, 10 ) );       // chamada da funcao, com parametros
// pausa para terminar
   printf ( "Pressionar ENTER para terminar." );
   getchar ( );
   return ( 0 );
} // fim do programa
```

Programa em C++:

```
// Exemplo 4.
// Calcular a soma dos 10 primeiros numeros naturais.
//
// bibliotecas necessárias
#include <iostream>
using namespace std;
//
// definir funcao com dois parametros
int SOMAR ( int LI, int LS )
{
 // 1. definir dados locais
   int X,                                  // numero natural
       SOMA = 0;                           // soma dos valores lidos
 // 2.gerar N numeros naturais
   for ( X = LI; X <= LS; X = X+1 )
   {
     // 2.1 somar cada numero natural
     SOMA = SOMA + X;
   } // fim do bloco repetitivo
   // retorno do valor calculado
   return ( SOMA );
} // fim SOMAR ( )
//
// parte principal
//
int main (void)
{
// calcular a soma dos 10 primeiros naturais
// 3. mostrar resultado
   cout << “\nSoma dos 10 primeiros naturais =“
        << SOMAR ( 1, 10 );           // chamada da funcao, com parametros
// pausa para terminar
   cout << "Pressionar ENTER para terminar.";
   getchar ( );
   return EXIT_SUCCESS;
} // fim do programa
```

Programa em C#:

```
/*
 * Exemplo 4
 * Calcular a soma dos 10 primeiros numeros naturais.
 */
using System;

class Exemplo_4
{
// definir funcao com parametros
   public static int SOMAR ( int LI, int LS )
   {
   // 1. definir dados locais
      int X,              // numero natural
          SOMA = 0; // soma dos valores lidos
   // 2. gerar N numeros naturais
      for ( X = LI; X <= LS; X = X+1 )
      {
      // 2.1. somar cada numero natural
         SOMA = SOMA + X;
      } // fim do bloco repetitivo
   // retorno do valor calculado
      return ( SOMA );
   } // fim SOMAR ( )
//
// parte principal
//
   public static void Main ( )
   {
   // 3. mostrar resultado
      Console.WriteLine ( "\nSoma dos 10 primeiros naturais = " +
                                 SOMAR( 1, 10 ) ); // chamada de funcao com parametros
   // pausa para terminar
      Console.Write ( "\nPressionar ENTER para terminar." );
      Console.ReadLine ( );
   } // end Main ( )

} // fim Exemplo_4 class
```

Programa em Java:

```
/**
  * Exemplo 4
  * Calcular a soma dos 10 primeiros numeros naturais.
  */

// ---------------------------------------------- classes necessarias

// ---------------------------------------------- definicao de classe

class Exemplo_4
{
// definir funcao com parametros
   public static int SOMAR ( int LI, int LS )
   {
   // 1. definir dados locais
      int X,                    // numero natural
          SOMA = 0;             // soma dos valores lidos
   // 2. gerar N numeros naturais
      for ( X = LI; X <= LS; X = X+1 )
      {
      // 2.1. somar cada numero natural
         SOMA = SOMA + X;
      } // fim do bloco repetitivo
   // retorno do valor calculado
      return ( SOMA );
   } // fim SOMAR ( )
//
// parte principal
//
   public static void main ( String [ ] args )
   {
   // 3. mostrar resultado
      System.out.println ( "\nSoma dos 10 primeiros naturais = " +
                           SOMAR( 1, 10 ) ); // chamada de funcao com parametros
   // pausa para terminar
      System.out.print ( "\nPressionar ENTER para terminar." );
      System.console( ).readLine( );
   } // end main ( )

} // fim Exemplo_4 class
```

Programa em Python:

```
# Exemplo 4a.
# Calcular a soma dos 10 primeiros numeros naturais.
#
# definir funcao com parametros
def SOMAR ( LI, LS ):
# parametros:
#int LI,  # limite inferior
#    LS;  # limite superior
#
# 1. definir dados locais
  X    = 0;          # numero natural
  SOMA = 0;      # soma dos valores lidos
# 2.gerar 10 numeros naturais
  for X in range ( LS, LI-1, -1 ):
  # 2.1 somar cada numero natural
    SOMA = SOMA + X;
  # fim do bloco repetitivo
# retorno do valor calculado
  return ( SOMA );
# SOMAR( )
#
# parte principal
#
# 3. mostrar resultado
#    chamada da funcao, com parametros
print ( "\nSoma dos 10 primeiros naturais = ", SOMAR( 1, 10 ) );
# pausa para terminar
print ( "\nPressionar ENTER para terminar.\n" );
input ( );
# fim do programa
```

Outra versão da função em Python, com variação também decrescente:

```
# definir funcao com parametros
def SOMAR ( LI, LS ):
# parametros:
#int LI,  # limite inferior
#    LS;  # limite superior
#
# 1. definir dados locais
   X    = 0;          # numero natural
   SOMA = 0;      # soma dos valores lidos
# 2.gerar 10 numeros naturais
   for X in reversed ( range ( LI, LS+1, 1 ) ):
   # 2.1 somar cada numero natural
      SOMA = SOMA + X;
   # fim do bloco repetitivo
# retorno do valor calculado
   return ( SOMA );
# SOMAR( )
```

Exercícios

1. Fazer um algoritmo para :
   - ler um número indeterminado de dados, contendo cada um, um valor inteiro (N);
   - o último valor será igual a zero e não deverá ser considerado;
   - calcular e mostrar a soma dos números naturais até o limite de cada valor lido (N).

2. Fazer um algoritmo para :
   - ler um número indeterminado de dados, contendo cada um, um valor inteiro (N);
   - o último valor será igual a zero e não deverá ser considerado;
   - ler também do teclado dois valores inteiros para indicar um intervalo fechado;
   - calcular e mostrar a soma dos (N) primeiros valores de um intervalo.

3. Fazer um algoritmo para :
   - ler um número indeterminado de dados, contendo cada um, um valor inteiro (N);
   - o último valor será igual a zero e não deverá ser considerado;
   - calcular e mostrar a soma dos (N) primeiros números pares.

4. Fazer um algoritmo para :
   - ler um número indeterminado de dados, contendo cada um, um valor inteiro (N);
   - o último valor será igual a zero e não deverá ser considerado;
   - calcular e mostrar a soma dos (N) primeiros números ímpares.

5. Fazer um algoritmo para :
   - ler um número indeterminado de dados, contendo cada um, um valor inteiro (N);
   - o último valor será igual a zero e não deverá ser considerado;
   - calcular e mostrar a soma dos quadrados dos (N) primeiros números naturais.

Exemplo 5.

Fazer um algoritmo para:
- implementar uma função para calcular o fatorial de um número inteiro;
- ler um número inteiro do teclado;
- calcular e mostrar o valor calculado.

Análise de dados:

- Dados da função:

Dado	Tipo	Valor Inicial	Obs.
N	inteiro		número de termos (parâmetro)
X	inteiro		contador
PRODUTO	inteiro	1	produto dos termos

- Fórmulas que relacionam os dados :

$$n\,! = \prod_{i=1}^{n} i = 1 \times 2 \times 3 \times \ldots \times (n-1) \times n$$

$0\,! = 1$ (por definição)

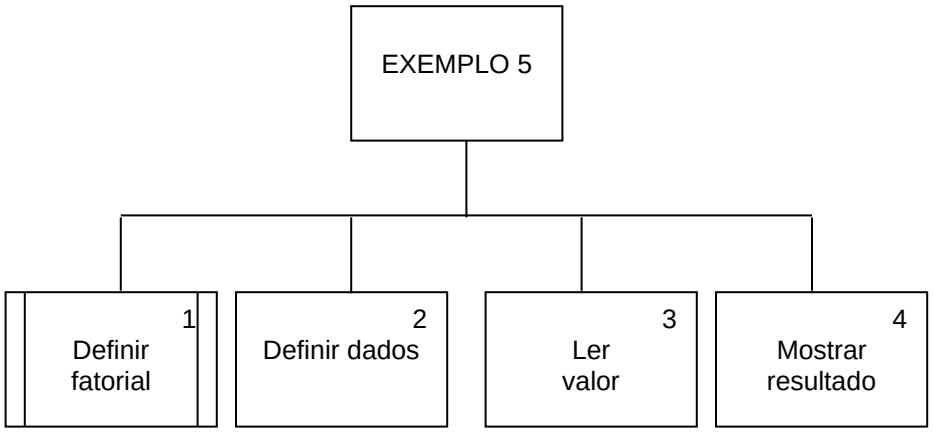
logo,

PRODUTO = PRODUTO * X; para cada número natural menor ou igual a N

- Dados da parte principal:

Dado	Tipo	Valor Inicial	Obs.
X	inteiro		valor lido

Diagrama funcional:



- Avaliação da solução :

5 ! = 5 * 4 * 3 * 2 * 1 = 120

Algoritmo, com função:

Esboço:

Primeira versão, só comentários.

Exemplo 5	v.1
Ação	Bloco
! definir função FATORIAL (inteiro N)	1
! definir dados da parte principal	2
! ler um valor inteiro	3
! mostrar resultado	4

Segunda versão, refinar o segundo bloco.

Exemplo 5	v.2
Ação	Bloco
! definir função FATORIAL (inteiro N)	1
! definir dados inteiro X; ! número inteiro	2
! ler um valor inteiro	3
! mostrar resultado	4

Terceira versão, refinar o terceiro e quarto blocos.

Exemplo 5	v.3
Ação	Bloco
! definir função FATORIAL (inteiro N)	1
! definir dados inteiro X; ! número inteiro	2
! ler um valor inteiro tela ← "\nQual o valor de X ?"; X ← teclado;	3
! mostrar resultado tela ← ("\nFatorial de ", X, " = ", FATORIAL (X));	4

Definição da função, no primeiro bloco.

Exemplo 5	v.4
Ação	Bloco
! definir função	1
inteiro função FATORIAL (inteiro N) ! com parâmetros	
! definir dados locais inteiro X, ! contador PRODUTO = 1; ! produto de naturais	1.1
! gerar números naturais em ordem decrescente	1.2
repetir para (X = N : -1 : 1) ! multiplicar cada número natural PRODUTO = PRODUTO * X; fim para ! a geração de N naturais	1.2.1
retornar (PRODUTO);	
! definir dados inteiro X; ! número inteiro	2
! ler um valor inteiro tela ← "\nQual o valor de X ?"; X ← teclado;	3
! mostrar resultado tela ← ("\nFatorial de ", X, " = ", FATORIAL (X));	4

Programa em SCILAB:

```
// Exemplo 5.
// Calcular o fatorial de um numero inteiro.
//
// definir funcao com parametro
function retorno = FATORIAL ( N )
// parametro:
// int N;                  // valor a ser calculado
//
// 1.1. definir dados locais
   X              = 0;     // contador
   PRODUTO = 1;   // produto de naturais
// 1.2. gerar N numeros naturais
   for ( X = N : -1 : 1 )
   // 1.2.1 multiplicar cada numero natural
      PRODUTO = PRODUTO * X;
   end // fim do bloco repetitivo
   // retorno do valor calculado
   retorno = PRODUTO;
endfunction // FATORIAL( )
//
// parte principal
//
// 2. definir dados
   X = 0;      // numero inteiro
// 3. ler um valor inteiro
   clc;   // limpar a area de trabalho
   X = input ( "\nQual o valor de X ? " );
// 4. mostrar resultado
   printf ( "\nFatorial de %d = %d", X, FATORIAL( X ) );
// pausa para terminar
   printf ( "\nPressionar ENTER para terminar.\n" );
   halt;
// fim do programa
```

Programa em C:

```
// Exemplo 5.
// Calcular o fatorial de um numero inteiro.
//
// bibliotecas necessarias
#include <stdio.h>
#include <stdlib.h>
//
// definir funcao com parametro
int FATORIAL ( int N )
{
 // 1.1. definir dados locais
   int X,                          // contador
       PRODUTO = 1;          // produto de naturais
 // 1.2. gerar N numeros naturais
   for ( X = N; X >= 1; X = X – 1 )
   {
     // 1.2.1 multiplicar cada numero natural
     PRODUTO = PRODUTO * X;
   } // fim do bloco repetitivo
   // retorno do valor calculado
   return ( PRODUTO );
} // fim FATORIAL( )
//
// parte principal
//
int main (void)
{
// 2. definir dados
   int X;                          // numero inteiro
// 3. ler um valor inteiro
   printf ( "\nQual o valor de X ?" );
   scanf ( "%d", &X );
// 4. mostrar resultado
   printf ( "\nFatorial de %d %s %d", X, " = ",
               FATORIAL( X ) );
// pausa para terminar
   printf ( "Pressionar ENTER para terminar." );
   getchar ( );
   return ( 0 );
} // fim do programa
```

Programa em C++:

```
// Exemplo 5.
// Calcular o fatorial de um numero inteiro.
//
// bibliotecas necessarias
#include <iostream>
using namespace std;
//
// definir funcao com parametro
int FATORIAL ( int N )
{
 // 1.1. definir dados locais
   int X,                          // contador
       PRODUTO = 1;          // produto de naturais
 // 1.2. gerar N numeros naturais
   for ( X = N; X >= 1; X = X – 1 )
   {
     // 1.2.1 multiplicar cada numero natural
     PRODUTO = PRODUTO * X;
   } // fim do bloco repetitivo
   // retorno do valor calculado
   return ( PRODUTO );
} // fim FATORIAL( )
//
// parte principal
//
int main (void)
{
// 2. definir dados
   int X;                          // numero inteiro
// 3. ler um valor inteiro
   cout << "\nQual o valor de X ?";
   cin   >>  X;
// 4. mostrar resultado
   cout << "\nFatorial de " << X << " = "
        << FATORIAL( X );
// pausa para terminar
   cout << "Pressionar ENTER para terminar.";
   cin.get ( );
   return EXIT_SUCCESS;
} // fim do programa
```

Programa em C#:

```
/*
 * Exemplo 5
 * Calcular o fatorial de uma numero inteiro.
 */
using System;

class Exemplo_5
{
// definir funcao com parametro
   public static int FATORIAL ( int N )
   {
   // 1.1. definir dados locais
      int X,                  // contador
          PRODUTO = 1; // produto de naturais
   // 1.2. gerar N numeros naturais
      for ( X = N; X >= 1; X = X-1 )
      {
      // 1.2.1. multiplicar cada numero natural
         PRODUTO = PRODUTO * X;
      } // fim do bloco repetitivo
   // retorno do valor calculado
      return ( PRODUTO );
   } // fim FATORIAL ( )
//
// parte principal
//
   public static void Main ( )
   {
   // 2. definir dados
      int X;                // numero inteiro
   // 3. ler um valor inteiro
      Console.Write ( "\nQual o valor de X ?" );
      X = int.Parse ( Console.ReadLine ( ) );
   // 4. mostrar resultado
      Console.WriteLine ( "\nFatorial de " + X + " = " +
                                 FATORIAL( X ) );
   // pausa para terminar
      Console.Write ( "\nPressionar ENTER para terminar." );
      Console.ReadLine ( );
   } // end Main ( )

} // fim Exemplo_5 class
```

Programa em Java:

```
/**
  * Exemplo 5
  * Calcular o fatorial de uma numero inteiro.
  */

// ------------------------------------------------ classes necessarias

// ------------------------------------------------ definicao de classe

class Exemplo_5
{
// definir funcao com parametro
   public static int FATORIAL ( int N )
   {
   // 1.1. definir dados locais
      int X,                        // contador
          PRODUTO = 1;              // produto de naturais
   // 1.2. gerar N numeros naturais
      for ( X = N; X >= 1; X = X-1 )
      {
      // 1.2.1. multiplicar cada numero natural
         PRODUTO = PRODUTO * X;
      } // fim do bloco repetitivo
   // retorno do valor calculado
      return ( PRODUTO );
   } // fim FATORIAL ( )
//
// parte principal
//
   public static void main ( String [ ] args )
   {
   // 2. definir dados
      int X;                        // numero inteiro
   // 3. ler um valor inteiro
      System.out.print ( "\nQual o valor de X ? " );
      X = Integer.parseInt ( System.console( ).readLine( ) );
   // 4. mostrar resultado
      System.out.println ( "\nFatorial de " + X + " = " +
                           FATORIAL( X ) );
   // pausa para terminar
      System.out.print ( "\nPressionar ENTER para terminar." );
      System.console( ).readLine( );
   } // end main ( )

} // fim Exemplo_5 class
```

Programa em Python:

```
# Exemplo 5.
# Calcular o fatorial de um numero inteiro.
#
# definir funcao com parametro
def FATORIAL ( N ):
# parametro:
# int N;                    # valor a ser calculado
#
# 1.1. definir dados locais
  X = 0;                    # contador
  PRODUTO = 1;              # produto de naturais
# 1.2. gerar N numeros naturais
#      em ordem decrescente (reversed)
  for X in range ( N, 1-1, -1 ):
  # 1.2.1 multiplicar cada numero natural
    PRODUTO = PRODUTO * X;
  # fim do bloco repetitivo
# retorno do valor calculado
  return ( PRODUTO );
# FATORIAL( )
#
# parte principal
#
# 2. definir dados
X = 0;                      # numero inteiro
# 3. ler um valor inteiro
X = int ( input ( "\nQual o valor de X ? " ) );
# 4. mostrar resultado
print  ( "\nFatorial de ", X, " = ", FATORIAL( X ) );
# pausa para terminar
print  ( "\nPressionar ENTER para terminar.\n" );
input ( );
# fim do programa
```

Exercícios

1. Escrever uma função para calcular o somatório abaixo :

$$S(n) = \frac{1}{0!} + \frac{1}{1!} + \frac{1}{2!} + \frac{1}{3!} + \quad \cdots + \frac{1}{n!}$$

recebendo o número de termos (N) como parâmetro.

2. Escrever uma função para calcular com 20 parcelas :

$$e^x = \frac{x^0}{0!} + \frac{x^1}{1!} + \frac{x^2}{2!} + \frac{x^3}{3!} + \quad \cdots$$

recebendo um valor real (X) como parâmetro.

3. Escrever uma função para gerar o cosseno de um ângulo (X), em radianos,

$$\cos(x,n) = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \quad \cdots$$

recebendo um valor real (X) como parâmetro, e
para (N) termos, com (N) lido do teclado.

4. Escrever uma função para gerar o seno de um ângulo (X), em radianos,

$$sen(x,n) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \quad \cdots$$

recebendo um valor real (X) como parâmetro, e
para (N) termos, com (N) lido do teclado.

5. Escrever uma função para calcular o somatório :

$$S(x) = \frac{x^{50}}{0!} - \frac{x^{49}}{1!} + \frac{x^{48}}{2!} - \quad \cdots \quad - \frac{x^1}{49!} + \frac{x^0}{50!}$$

recebendo um valor real (X) como parâmetro.

Exemplo 6.

Fazer um algoritmo para:
- implementar uma função para calcular a distância entre dois pontos no plano;
- ler as coordenadas do primeiro ponto do teclado (X1, Y1);
- ler as coordenadas do segundo ponto do teclado (X2, Y2);
- calcular e mostrar o valor calculado.

Análise de dados:

- Dados da função:

Dado	Tipo	Valor Inicial	Obs.
X1	real		coordenadas do primeiro ponto
Y1	real		
X2	real		coordenadas do segundo ponto
Y2	real		
d	real		distância entre pontos

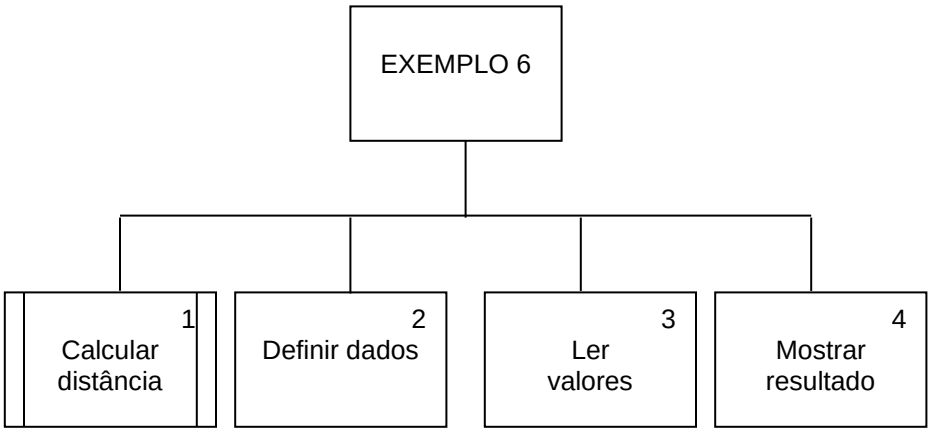
- Fórmulas que relacionam os dados :

$$d = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

- Dados da parte principal:

Dado	Tipo	Valor Inicial	Obs.
X1	real		coordenadas do primeiro ponto
Y1	real		
X2	real		coordenadas do segundo ponto
Y2	real		

Diagrama funcional:



- Avaliação da solução :

(X1, Y1) = (1,5) (X2,Y2) = (4,9) d = 5 [m]

Algoritmo, com função:

Esboço:

Primeira versão, só comentários.

Exemplo 6	v.1
Ação	Bloco
! definir função DISTÂNCIA ! (real X1, real Y1, real X2, real Y2)	1
! definir dados da parte principal	2
! ler valores dos pontos	3
! mostrar resultado	4

Segunda versão, refinar o segundo bloco.

Exemplo 6	v.2
Ação	Bloco
! definir função	1
real DISTANCIA (real X1, real Y1, real X2, real Y2)	

! definir dados locais
  real X1, Y1, ! coordenadas do primeiro ponto
  X2, Y2; ! coordenadas do segundo ponto
	2

! ler valores dos pontos	3
! mostrar resultado	4

Terceira versão, refinar o terceiro e quarto blocos.

Exemplo 6	v.3
Ação	Bloco
! definir função	1
real DISTANCIA (real X1, real Y1, real X2, real Y2)	

! definir dados locais
  real X1, Y1, ! coordenadas do primeiro ponto
  X2, Y2; ! coordenadas do segundo ponto
	2

! ler valores dos pontos
  tela ← "\nQual o valor de X1 ?"; X1 ← teclado;
  tela ← "\nQual o valor de Y1 ?"; Y1 ← teclado;
  tela ← "\nQual o valor de X2 ?"; X2 ← teclado;
  tela ← "\nQual o valor de Y2 ?"; Y2 ← teclado;
	3

! mostrar resultado tela ← ("\nDistancia = ", DISTANCIA(X1,Y1,X2,Y2));	4

Definição da função, no primeiro bloco.

Exemplo 6	v.4
Ação	Bloco
! definir função	1
real DISTANCIA (real X1, real Y1, real X2, real Y2)	
! definir dados locais real D; ! distância	1.1
! calcular a distância	1.2
D = raiz((X1-X2)^2 + (Y1-Y2)^2);	1.2.1
retornar (D);	
! definir dados locais real X1, Y1, ! coordenadas do primeiro ponto X2, Y2; ! coordenadas do segundo ponto	2
! ler valores dos pontos tela ← "\nQual o valor de X1 ?"; X1 ← teclado; tela ← "\nQual o valor de Y1 ?"; Y1 ← teclado; tela ← "\nQual o valor de X2 ?"; X2 ← teclado; tela ← "\nQual o valor de Y2 ?"; Y2 ← teclado;	3
! mostrar resultado tela ← ("\nDistancia = ", DISTANCIA(X1,Y1,X2,Y2));	4

Programa em SCILAB:

```
// Exemplo 6.
// Calcular a distancia entre dois pontos no plano.
//
// definir funcao com parametros
function retorno = DISTANCIA ( X1, Y1, X2, Y2 )
// parametro:
//      float X1, Y1;      // dados do primeiro ponto
//      float X2, Y2;      // dados do segundo ponto
//
// 1.1. definir dados locais
   D = 0;                  // distancia
// 1.2. calcular a distancia
    D = sqrt ( (X1-X2) ^ 2 + (Y1-Y2) ^ 2 );
// retorno do valor calculado
    retorno = D ;
endfunction // DISTANCIA( )
//
// parte principal
//
// calcular a distancia entre dois pontos no plano
// 2. definir dados
   X1 = 0; Y1 = 0;      // coordenadas do primeiro ponto
   X2 = 0; Y2 = 0;      // coordenadas do segundo ponto
// 3. ler valores dos pontos
   clc;   // limpar a area de trabalho
   X1 = input ( "\nQual o valor de X1 ? " );
   Y1 = input ( "\nQual o valor de Y1 ? " );
   X2 = input ( "\nQual o valor de X2 ? " );
   Y2 = input ( "\nQual o valor de X2 ? " );
// 4. mostrar resultado
   printf ( "\nDistancia = %f" , DISTANCIA( X1,Y1,X2,Y2 ) );
// pausa para terminar
   printf ( "\nPressionar ENTER para terminar.\n" );
   halt;
// fim do programa
```

Programa em C:

```
// Exemplo 6.
// Calcular a distancia entre dois pontos no plano.
//
// bibliotecas necessarias
#include <stdio.h>
#include <stdio.h>
#include <math.h>
//
// definir funcao com parametros
float DISTANCIA ( float X1, float Y1, float X2, float Y2 )
{
 // 1.1. definir dados locais
   float D;                          // distancia
// 1.2. calcular a distancia
   D = sqrt ( pow ((X1-X2),2) + pow ((Y1-Y2),2) );
// retorno do valor calculado
   return ( D );
} // fim DISTANCIA( )
//
// parte principal
//
int main (void)
{
// 2. definir dados
   float X1, Y1,                     // coordenadas do primeiro ponto
         X2, Y2,                    // coordenadas do segundo ponto
// 3. ler valores dos pontos
   printf ( "\nQual o valor de X1 ? " );   scanf ( "%f", &X1 );
   printf ( "\nQual o valor de Y1 ? " );   scanf ( "%f", &Y1 );
   printf ( "\nQual o valor de X2 ? " );   scanf ( "%f", &X2 );
   printf ( "\nQual o valor de Y2 ? " );   scanf ( "%f", &Y2 );
// 4. mostrar resultado
   printf ( "\nDistancia = %f",
               DISTANCIA (X1,Y1,X2,Y2) );
// pausa para terminar
   pirntf ( "Pressionar ENTER para terminar." );
   getchar ( );
   return ( 0 );
} // fim do programa
```

Programa em C++:

```
// Exemplo 6.
// Calcular a distancia entre dois pontos no plano.
//
// bibliotecas necessarias
#include <iostream>
using namespace std;
//
// definir funcao com parametros
double DISTANCIA ( float X1, float Y1, float X2, float Y2 )
{
 // 1.1. definir dados locais
   double D;                        // distancia
// 1.2. calcular a distancia
   D = sqrt ( pow ((X1-X2),2) + pow ((Y1-Y2),2) );
// retorno do valor calculado
   return ( D );
} // fim DISTANCIA( )
//
// parte principal
//
int main (void)
{
// 2. definir dados
   double X1, Y1,                // coordenadas do primeiro ponto
          X2, Y2,               // coordenadas do segundo ponto
// 3. ler valores dos pontos
   cout << "\nQual o valor de X1 ? ";   cin >> X1;
   cout << "\nQual o valor de Y1 ? ";   cin >> Y1;
   cout << "\nQual o valor de X2 ? ";   cin >> X2;
   cout << "\nQual o valor de Y2 ? ";   cin >> Y2;
// 4. mostrar resultado
   cout << "\nDistancia = "
        <<      DISTANCIA (X1,Y1,X2,Y2);
// pausa para terminar
   cout << "Pressionar ENTER para terminar.";
   cin.get ( );
   return EXIT_SUCCESS;
} // fim do programa
```

Programa em C#:

```
/*
 * Exemplo 6
 * Calcular a distancia entre dois pontos no plano.
 */
using System;

class Exemplo_6
{
// definir funcao com parametros
   public static double DISTANCIA ( double X1, double Y1, double X2, double Y2 )
   {
   // 1.1. definir dados locais
      double D;        // distancia
   // 1.2. calcular a distancia
      D = Math.Sqrt ( Math.Pow ((X1-X2), 2) + Math.Pow ((Y1-Y2), 2));
   // retorno do valor calculado
      return ( D );
   } // fim DISTANCIA ( )
//
// parte principal
//
   public static void Main ( )
   {
   // 2. definir dados
      double X1, Y1,   // coordenadas do primeiro ponto
             X2, Y2;   // coordenadas do segundo  ponto
   // 3. ler valores dos pontos
      Console.Write ( "\nQual o valor de X1 ? "   );
      X1 = double.Parse ( Console.ReadLine ( ) );
      Console.Write ( "\nQual o valor de Y1 ? "   );
      Y1 = double.Parse ( Console.ReadLine ( ) );
      Console.Write ( "\nQual o valor de X2 ? "   );
      X2 = double.Parse ( Console.ReadLine ( ) );
      Console.Write ( "\nQual o valor de Y2 ? "   );
      Y2 = double.Parse ( Console.ReadLine ( ) );
   // 4. mostrar resultado
      Console.WriteLine ( "\nDistancia = " +
                                  DISTANCIA( X1, Y1, X2, Y2 ) );
   // pausa para terminar
      Console.Write ( "\nPressionar ENTER para terminar." );
      Console.ReadLine ( );
   } // end Main ( )

} // fim Exemplo_6 class
```

Programa em Java:

```
/**
  * Exemplo 6
  * Calcular a distancia entre dois pontos no plano.
  */

// ------------------------------------------------ classes necessarias

// ------------------------------------------------ definicao de classe

class Exemplo_6
{
// definir funcao com parametros
   public static double DISTANCIA ( double X1, double Y1, double X2, double Y2 )
   {
   // 1.1. definir dados locais
      double D;                              // distancia
   // 1.2. calcular a distancia
      D = Math.sqrt ( Math.pow ((X1-X2), 2) + Math.pow ((Y1-Y2), 2));
   // retorno do valor calculado
      return ( D );
   } // fim DISTANCIA ( )
//
// parte principal
//
   public static void main ( String [ ] args )
   {
   // 2. definir dados
      double X1, Y1,             // coordenadas do primeiro ponto
             X2, Y2;             // coordenadas do segundo  ponto
   // 3. ler valores dos pontos
      System.out.print ( "\nQual o valor de X1 ? " );
      X1 = Double.parseDouble ( System.console( ).readLine( ) );
      System.out.print ( "\nQual o valor de Y1 ? " );
      Y1 = Double.parseDouble ( System.console( ).readLine( ) );
      System.out.print ( "\nQual o valor de X2 ? " );
      X2 = Double.parseDouble ( System.console( ).readLine( ) );
      System.out.print ( "\nQual o valor de Y2 ? " );
      Y2 = Double.parseDouble ( System.console( ).readLine( ) );
   // 4. mostrar resultado
      System.out.println ( "\nDistancia = " +
                           DISTANCIA( X1, Y1, X2, Y2 ) );
   // pausa para terminar
      System.out.print ( "\nPressionar ENTER para terminar." );
      System.console( ).readLine( );
   } // end main ( )

} // fim Exemplo_6 class
```

Programa em Python:

```
# Exemplo 6.
# Calcular a distancia entre dois pontos no plano.

from math import sqrt

#
# definir funcao com parametros
def DISTANCIA ( X1, Y1, X2, Y2 ):
# parametro:
# float X1, Y1;   # dados do primeiro ponto
# float X2, Y2;   # dados do segundo ponto
#
# 1.1. definir dados locais
   D = 0;# distancia
# 1.2. calcular a distancia
   D = sqrt ( (X1-X2) ** 2 + (Y1-Y2) ** 2 );
# retorno do valor calculado
   return ( D );
# DISTANCIA( )
#
# parte principal
#
# calcular a distancia entre dois pontos no plano
# 2. definir dados
X1 = 0; Y1 = 0;  # coordenadas do primeiro ponto
X2 = 0; Y2 = 0;  # coordenadas do segundo ponto
# 3. ler valores dos pontos
X1 = float ( input ( "\nQual o valor de X1 ? " ) );
Y1 = float ( input ( "\nQual o valor de Y1 ? " ) );
X2 = float ( input ( "\nQual o valor de X2 ? " ) );
Y2 = float ( input ( "\nQual o valor de X2 ? " ) );
# 4. mostrar resultado
print  ( "\nDistancia = " , DISTANCIA( X1,Y1,X2,Y2 ) );
# pausa para terminar
print  ( "\nPressionar ENTER para terminar.\n" );
input ( );
# fim do programa
```

Exercícios

1. Escrever uma função para calcular a área de um triângulo dada a base e altura.

2. Escrever uma função para calcular a área de um retângulo dados os lados.

3. Escrever uma função para calcular a área de um trapézio dadas a base e a altura.

4. Escrever uma função para calcular o volume de uma esfera dado o raio.

5. Escrever uma função para calcular o comprimento de uma circunferência de raio.

6. Escrever uma função para calcular uma área circular dado o diâmetro.

Exemplo 7.

Fazer um algoritmo para:
- implementar uma função para verificar se um número real é nulo ou não;
- ler vários números do teclado e testar cada um deles com esta função.

Análise de dados:

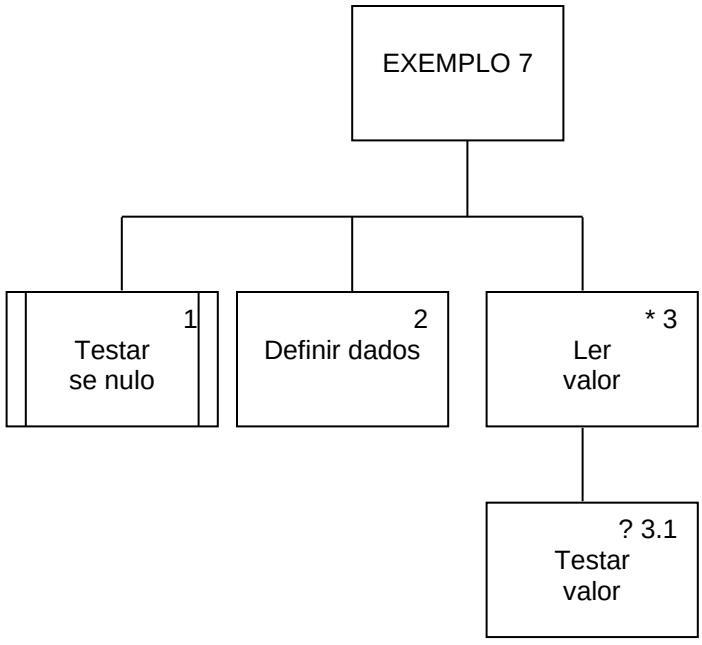
- Dados da função para teste de nulo:

Dado	Tipo	Valor Inicial	Obs.
X	real		valor lido
Resposta	inteiro		resultado da verificação

- Dados da parte principal:

Dado	Tipo	Valor Inicial	Obs.
X	real		valor lido
Resposta	caractere		controle da repetição

Diagrama funcional:



- Avaliação da solução :

X = -1 X = 0 X = -1

Algoritmo, com função:

Esboço:

Primeira versão, só comentários.

Exemplo 7	v.1
Ação	Bloco
! definir função NULO (real X)	1
! definir dados da parte principal	2
! ler valores dos pontos	3
! mostrar resultado	4

Segunda versão, refinar o segundo bloco.

Exemplo 7	v.2
Ação	Bloco
! definir função	1
inteiro função NULO (real X)	
! definir dados real X; ! valor a ser lido	2
! ler valores	3
! testar valor	3.1

Terceira versão, refinar o terceiro bloco.

Exemplo 7			v.3
Ação			Bloco
! definir função			1
inteiro função NULO (real X)			
! definir dados real X; ! valor a ser lido			2
! ler valores			3
tela ← "\nQual o valor de X ?"; X ← teclado;			
! testar valor			3.1
NULO(X)?	V	tela ← "\nValor nulo.";	
	F	tela ← "\nValor diferente de zero.";	
enquanto houver dados			

Definição da função, no primeiro bloco.

Exemplo 7	v.3
Ação	Bloco
! definir função	1
inteiro NULO (real X)	
inteiro Resposta; se (X = 0) { Resposta = 1; } senão { Resposta = 0; } retornar (Resposta);	
	2
! definir dados real X; ! valor a ser lido	
! ler valores	3
tela ← "\nQual o valor de X ?"; X ← teclado;	
! testar valor	3.1
NULO(X)? V: tela ← "\nValor nulo.";	
NULO(X)? F: tela ← "\nValor diferente de zero.";	
enquanto houver dados	

Definição da função, no terceiro bloco.

Exemplo 7	v.3
Ação	Bloco
! definir função	1
inteiro NULO (real X)	
inteiro Resposta; se (X = 0) Resposta = 1; else Resposta = 0; retornar (Resposta);	
	2
! definir dados real X; ! valor a ser lido caractere Resposta; ! controle da repetição	
! ler valores	3
tela ← "\nQual o valor de X ?"; X ← teclado;	
! testar valor	3.1
se (NULO(X)) tela ← "\nValor nulo."; senão tela ← "\nValor diferente de zero.";	
tela ← "\nQuer continuar (S,N) ? “; Resposta ← teclado;;	
enquanto Resposta = “S”;	

Programa em SCILAB:

```
// Exemplo 7.
// Calcular a distancia entre dois pontos no plano.
//
// definir funcao com parametros
function retorno = NULO ( X )
// parametro:
// float X;              // valor a ser testado
//
// definir dados locais
// int Resposta;
  if ( X == 0 )
    Resposta = 1;
  else
    Resposta = 0;
  end // fim se nulo
  retorno = Resposta;
endfunction // NULO( )
//
// parte principal
//
// testar se valores lidos são nulos
// 2. definir dados
// definir dados
  X = 0;                // valor a ser lido
  Resposta = 1;         // controle da repeticao
// 3. ler valores
  clc;                  // limpar a area de trabalho
  while ( Resposta ~= 0 )
    X = input ( "\nQual o valor de X ? " );
  // 3.1. testar valor
    if ( NULO( X ) )
      printf ( "\nValor nulo." );
    else
      printf ( "\nValor diferente de zero." );
    end // if
    Resposta = input ( "\nQuer continuar (Sim=1/Nao=0) ? " );
  end // while
// pausa para terminar
  printf ( "\nPressionar ENTER para terminar.\n" );
  halt;
// fim do programa
```

Programa em C++:

```
// Exemplo 7.
// Calcular a distancia entre dois pontos no plano.
//
// bibliotecas necessarias
#include <stdio.h>
#include <stdlib.h>
//
// definir funcao com parametros
int NULO ( float X )
{
 // definir dados locais
   int Resposta;
   if ( X == 0 )
   { Resposta = 1; }
   else
   { Resposta = 0; }
   return( Resposta );
} // fim NULO( )
//
// parte principal
//
int main (void)
{
// testar se valores lidos sao nulos
// 2. definir dados
// definir dados
   float X;                    // valor a ser lido
   char Resposta;              // controle da repeticao
// 3. ler valores
   do
   {
   // 3.1. ler um valor do teclado
      printf ( "\nQual o valor de X ? " );
      scanf ( "%f", &X );
   // 3.2. testar valor lido
      if ( NULO ( X ) )
      {   printf ( "\nValor nulo." );   }
      else
      {   printf ( "\nValor diferente de zero." );   }
   // 3.3. controlar a repeticao
      printf ( "\nQuer continuar (S/N) ? " );
      fflush ( stdin );           // limpar a entrada de dados
      scanf ( "%c", &Resposta );
    }
   while ( Resposta == 'S' ); // pausa para terminar
   printf ( "Pressionar ENTER para terminar." );
   getchar ( );
   return ( 0 );
} // fim do programa
```

Programa em C++:

```
// Exemplo 7.
// Calcular a distancia entre dois pontos no plano.
//
// bibliotecas necessarias
#include <iostream>
using namespace std;
//
// definir funcao com parametros
int NULO ( float X )
{
 // definir dados locais
   int Resposta;
   if ( X == 0 )
   { Resposta = 1; }
   else
   { Resposta = 0; }
   return( Resposta );
} // fim NULO( )
//
// parte principal
//
int main (void)
{
// testar se valores lidos sao nulos
// 2. definir dados
// definir dados
   double X;                    // valor a ser lido
   char Resposta;               // controle da repeticao
// 3. ler valores
   do
   {
   // 3.1. ler um valor do teclado
      cout << "\nQual o valor de X ? ";
      cin  >> X;
   // 3.2. testar valor lido
      if ( NULO ( X ) )
      {   cout << "\nValor nulo.";   }
      else
      {   cout << "\nValor diferente de zero.";   }
   // 3.3. controlar a repeticao
      cout << "\nQuer continuar (S/N) ? ";
      cin   >> Resposta;        // Resposta = cin.get ( );
    }
   while ( Resposta == 'S' ); // pausa para terminar
   cout << "Pressionar ENTER para terminar.";
   cin.get ( );
   return EXIT_SUCCESS;
} // fim do programa
```

Programa em C#:

```
/**
  * Exemplo 7
  * Testar se valores lidos sao nulos.
  */
using System;

class Exemplo_7
{
// definir funcao com parametro
   public static boolean NULO ( double X )
   {
   // 1.1. definir dados locais
      boolean Resposta;
   // 1.2. testar valor
      if ( X == 0.0 )
      { Resposta = true;  }
      else
      { Resposta = false; }
   // retorno da resposta
      return ( Resposta );
   } // fim NULO ( )
//
// parte principal
//
   public static void Main ( )
   {
   // 2. definir dados
      double X;          // valor a ser lido
      char   Resposta; // controle da repeticao
   // 3. ler valores
      do
      {
      // 3.1. ler um valor do teclado
         Console.WriteLine ( "\nQual o valor de X ?" );
         X = double.Parse ( Console.ReadLine ( ) );
      // 3.2. testar valor lido
         if ( NULO ( X ) )
         { Console.WriteLine ( "\nValor nulo." ); }
         else
         { Console.WriteLine ( "\nValor diferente de zero." );  }
      // 3.3. controlar a repeticao
         Console.Write ( "\nQuer continuar (S/N) ? " );
         Resposta = (char) Console.Read ( );
         Console.ReadLine ( );   // limpar o buffer
      }
      while ( Resposta == 'S' );
   // pausa para terminar
      Console.Write ( "\nPressionar ENTER para terminar." );
      Console.ReadLine ( );
   } // end Main ( )

} // fim Exemplo_7 class
```

Programa em Java:

```
/**
  * Exemplo 7
  * Testar se valores lidos sao nulos.
  */
// ------------------------------------------------ classes necessarias

// ------------------------------------------------ definicao de classe

class Exemplo_7
{
// definir funcao com parametro
   public static boolean NULO ( double X )
   {
   // 1.1. definir dados locais
      boolean Resposta;
   // 1.2. testar valor
      if ( X == 0.0 )
      { Resposta = true;  }
      else
      { Resposta = false; }
   // retorno da resposta
      return ( Resposta );
   } // fim NULO ( )
//
// parte principal
//
   public static void main ( String [ ] args )
   {
   // 2. definir dados
      double X;                    // valor a ser lido
      char   Resposta; // controle da repeticao
   // 3. ler valores
      do
      {
      // 3.1. ler um valor do teclado
         System.out.print ( "\nQual o valor de X ? " );
         X = Double.parseDouble ( System.console( ).readLine( ) );
      // 3.2. testar valor lido
         if ( NULO ( X ) )
         { System.out.println ( "\nValor nulo." ); }
         else
         { System.out.println ( "\nValor diferente de zero." );  }
      // 3.3. controlar a repeticao
         System.out.print ( "\nQuer continuar (S/N) ? " );
         Resposta = (System.console( ).readLine( )).charAt(0);
      }
      while ( Resposta == 'S' );
   // pausa para terminar
      System.out.print ( "\nPressionar ENTER para terminar." );
      System.console( ).readLine( );
   } // end main ( )
} // fim Exemplo_7 class
```

Programa em Python:

```
# Exemplo 7.
# Calcular a distancia entre dois pontos no plano.
#
# definir funcao com parametros
def NULO ( X ):
# parametro:
# float X;                    # valor a ser testado
#
# definir dados locais
# int Resposta;
  if ( X == 0 ):
    Resposta = 1;
  else:
    Resposta = 0;
# fim se nulo
 return ( Resposta );
# NULO( )
#
# parte principal
#
# testar se valores lidos sao nulos
# 2. definir dados
# definir dados
X = 0;                  # valor a ser lido
Resposta = 1;   # controle da repeticao
# 3. ler valores
while ( Resposta != 0 ):
  X = float ( input ( "\nQual o valor de X ? " ) );
# 3.1. testar valor
  if ( NULO( X ) ):
    print ( "\nValor nulo." );
  else:
    print ( "\nValor diferente de zero." );
  # fim se
  Resposta = int ( input ( "\nQuer continuar (Sim=1/Nao=0) ? " ) );
# enquanto ( Resposta != 0 )
# pausa para terminar
print ( "\nPressionar ENTER para terminar.\n" );
input ( );
# fim do programa
```

Procedimento

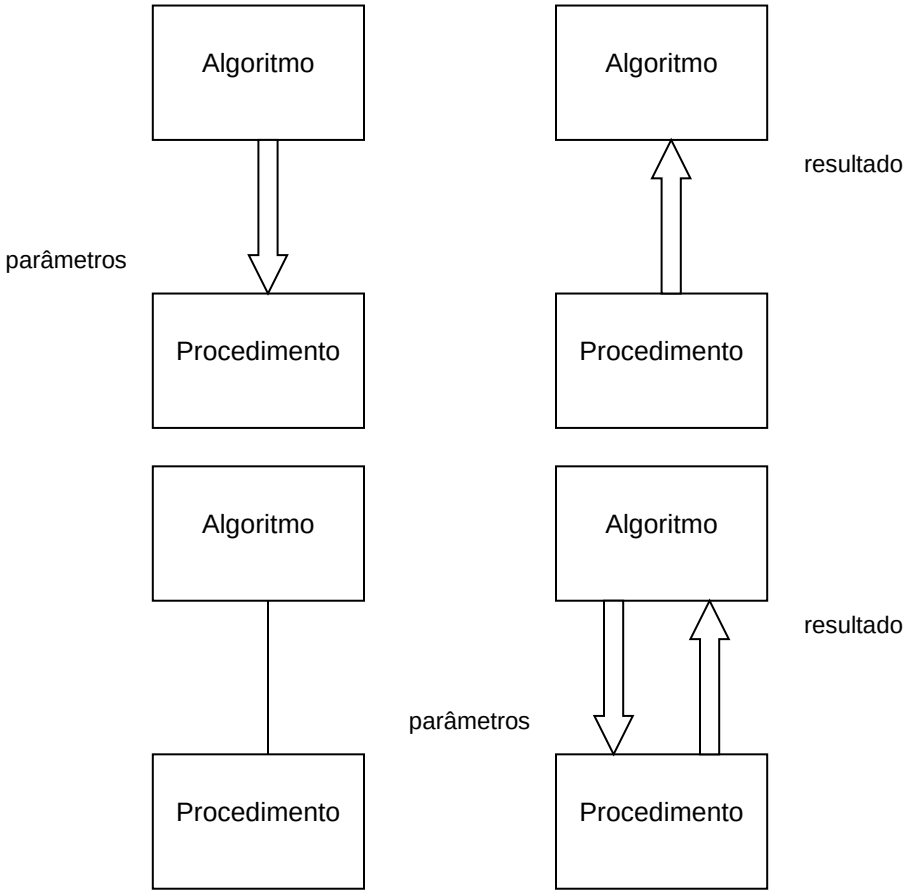
A definição de um procedimento, assim como a de uma função, deve preceder qualquer outro comando executável.

Um procedimento só tem suas ações executadas quando é feita uma chamada, que se resume no uso do nome do procedimento; acompanhada, ou não, da lista de parâmetros, conforme o caso.

Os procedimentos são normalmente empregados para substituir blocos de comandos.

Os procedimentos podem ter qualquer um dos relacionamentos mostrados abaixo, permitindo uma flexibilidade maior que a oferecida pelas funções, destacando-se, porém, que não são capazes de retornar qualquer resultado em seu próprio nome, apesar da expansão e substituição ocorrerem da mesma forma. O retorno de um ou mais resutlados poderá ocorrer se houver mecanismos para passagens de parâmetros por referência, por exemplo.

- Diagramas de relacionamento :



- Procedimentos sem parâmetros

A forma mais simples de utilização deste recurso é supor o encapsulamento de comandos, com definição de um determinado valor, equivalente a uma atribuição.

Exemplo:

```
início
! algoritmo para mostrar o valor do número (e)
! definir dado
   real E = 2.71828;
! mostrar o valor de (E)
  tela ← ("\ne = ", E);
fim.
```

Utilizando o conceito de procedimento, esta atribuição de valor poderia ser feita apenas encapsulando comandos.

Exemplo:

```
! definir dado global
  real E;

! definir procedimento
  procedimento NUMERO_E( )
     E = 2.71828;
  fim procedimento ! NUMERO_E( )

! parte principal
início
! algoritmo para mostrar o valor do numero (e)
! definir dado
  NUMERO_E( );            ! chamada do procedimento para atribuir o valor
! mostrar valor de ( E )
  tela ← ( "\ne = ", E );
fim.
```

Devido à restrição imposta, da variável ser visível tanto para o procedimento NUMERO_E quanto para a parte principal, é preciso manter a sua definição antes de ambos. Configura-se o uso de uma variável ***global.***.

Exemplo em SCILAB:

```
// definir dado global
    global E;
//
// parte principal
//
// algoritmo para mostrar o valor do numero (e)
// definir dado
    NUMEROE;              // chamada do procedimento para atribuir o valor
// mostrar valor de ( E )
    printf ( “\ne = %f” , E );
// fim do programa
```

Em um arquivo separado (NUMEROE.m):

```
// definir procedimento
    function NUMEROE
//
      global E;
//
       E = 2.71828;
// fim NUMEROE( )
```

Exemplo em C:

```
// definir dado global
   float E;

// definir procedimento
   void NUMERO_E ( void )
   {
     E = 2.71828;
   } // fim NUMERO_E ( )

// parte principal
   int main (void)
   {
   // algoritmo para mostrar o valor do numero (e)
   // definir dado
      NUMERO_E( );        // chamada do procedimento para atribuir o valor
   // mostrar valor de ( E )
      printf ( "\ne = %f", E );
      return ( 0 );
} // fim do programa
```

Exemplo em C++:

```
// definir dado global
   float E;

// definir procedimento
   void NUMERO_E ( void )
   {
     E = 2.71828;
   } // fim NUMERO_E ( )

// parte principal
   int main (void)
   {
   // algoritmo para mostrar o valor do numero (e)
   // definir dado
      NUMERO_E( );        // chamada do procedimento para atribuir o valor
   // mostrar valor de ( E )
      cout << "\ne = " << E ;
      return EXIT_SUCCESS;
} // fim do programa
```

Exemplo em C#:

```
class Exemplo
{
// 1. definir de dado global (e)
   public static double E; // valor de (e) com (x) termos
//
// 2. definir procedimento com parametro
   public static void NUMERO_E ( )
   {
      E = 2.71828;
   } // fim NUMERO_E ( )

// parte principal
   public static void main ( String [ ] args )
   {
  // algoritmo para mostrar o valor do numero (e)
  // definir dado
     NUMERO_E ( );        // chamada do procedimento para atribuir o valor
  // mostrar valor de ( E )
     System.out.println ( "\ne = " + E );
   } // end main ( )

} // fim Exemplo class
```

Exemplo em Java:

```
class Exemplo
{
// 1. definir de dado global (e)
   public static double E;        // valor de (e) com (x) termos
//
// 2. definir procedimento com parametro
   public static void NUMERO_E ( )
   {
      E = 2.71828;
   } // fim NUMERO_E ( )

// parte principal
   public static void main ( String [ ] args )
   {
  // algoritmo para mostrar o valor do numero (e)
  // definir dado
     NUMERO_E( );          // chamada do procedimento para atribuir o valor
  // mostrar valor de ( E )
     System.out.println ( "\ne = " + E );
   } // end main ( )

} // fim Exemplo class
```

Exemplo em Python:

```
# definir dado global
E = 0;
#
# definir procedimento com parametro
def NUMERO_E ( ):
      global E;
      E = 2.71828;
# fim NUMERO_E ( )
#
# parte principal
#
# algoritmo para mostrar o valor do numero (e)
# definir dado
NUMERO_E( );              # chamada do procedimento para atribuir o valor
# mostrar valor de ( E )
print  ( "\ne = " , E );
input ( );
# fim do programa
```

- Procedimentos com parâmetros

A utilização de parâmetros também permitirá uma flexibilidade ainda maior no uso de procedimentos.

Exemplo 8:

Calcular e mostrar o valor do número (e), com aproximações de 10 e 20 termos da soma:

$$e = 1 + \frac{1}{1!} + \frac{1}{2!} + \frac{1}{3!} + \quad \cdots$$

Análise de dados:

- Dados do procedimento:

Dado	Tipo	Valor Inicial	Obs.
N	inteiro		número de termos (parâmetro)
X	inteiro		contador
SOMA	real	1.0	somatório de (N) termos
PRODUTO	inteiro	1	produto dos termos

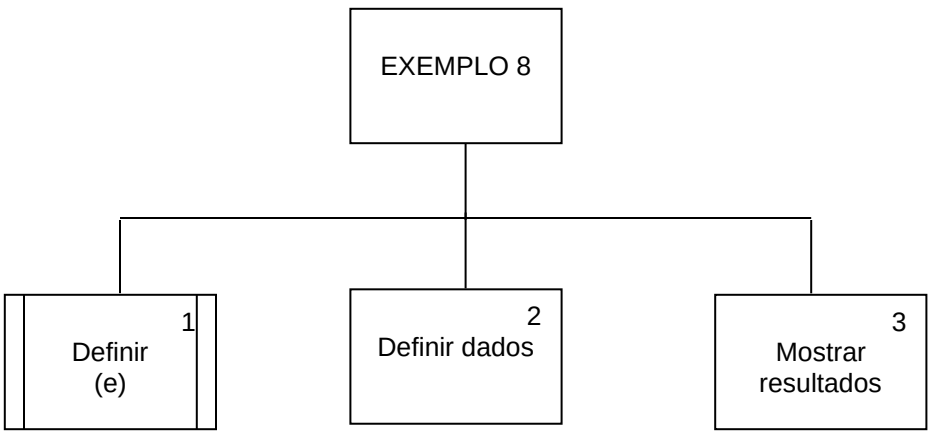
- Fórmulas que relacionam os dados :

PRODUTO = PRODUTO * X; para cada número natural menor ou igual a N
SOMA = SOMA + 1/PRODUTO; para cada produto calculado

- Dados da parte principal:

Dado	Tipo	Valor Inicial	Obs.
E	real		valor do número (e)

Diagrama funcional:



Algoritmo, com procedimento:

Esboço:

Primeira versão, só comentários.

Exemplo 8	v.1
Ação	Bloco
! definir procedimento EXP (inteiro N)	1
! definir dados da parte principal	2
! mostrar resultado	3

Segunda versão, só comentários.
Não haverá dados na parte principal, mas será necessário uma variável global (e).

Exemplo 8	v.2
Ação	Bloco
! definir dado global (e)	1
! definir procedimento EXP1 (inteiro N)	2
! mostrar resultado	3

Terceira versão, refinar o terceiro e quarto blocos.

Exemplo 8	v.3
Ação	Bloco
! definir dado global (e)	1
! definir procedimento EXP1 (inteiro N)	2
! mostrar resultado EXP1(10); tela ← (“\ne com 10 termos = “, E); EXP1(20); tela ← (“\ne com 20 termos = “, E);	3

Quarta versão, refinar o primeiro bloco.

Exemplo 8	v.4
Ação	Bloco
! definir dado global (e) real E; ! valor de (e) com (x) termos	1
! definir procedimento EXP1 (inteiro N)	2
! parte principal	
! mostrar resultado EXP1(10); tela ← (“\ne com 10 termos = “, E); EXP1(20); tela ← (“\ne com 20 termos = “, E);	3

Quinta versão, refinar o segundo bloco.

Exemplo 8	v.5
Ação	Bloco
! definir dado global (e) real E; ! valor de (e) com (x) termos	1
! definir procedimento EXP1 (inteiro N)	
procedimento EXP1 (inteiro N)	

! definir dados locais
  inteiro I, ! contador
  PRODUTO = 1; ! produto de naturais
  real SOMA = 1.0, ! soma de inversos
	2.1

! gerar (N-1) naturais em ordem crescente	2.2

repetir para (X = 1 : (N-1) : 1)
! calcular o fatorial
  PRODUTO = PRODUTO * X;
! acumular a soma dos inversos
  SOMA = SOMA + 1.0/PRODUTO;
fim repetir ! geração de (N-1) naturais	

! definição do valor de retorno E = SOMA;	2.3
! parte principal	

! mostrar resultado
  EXP1(10);
  tela ← (“\ne com 10 termos = “, E);
  EXP1(20);
  tela ← (“\ne com 20 termos = “, E);
	3

A substituição de parâmetros ocorre no momento da chamada, da mesma forma que na função, embora seja mantida a variável global para o valor de retorno.

Dessa forma, dentro do procedimento haverá apenas uma referência a uma variável global, e esta poderá ser alterada no momento da chamada do procedimento. Neste caso, não será feita apenas uma substituição de valor, o parâmetro ficará associado à variável enquanto durar a execução do procedimento, desligando-se quando ela tiver terminado, mas a variável permanecerá com o valor que lhe foi atribuído.

Observação :

Supor que a substituição de parâmetros é feita da esquerda para a direita, avaliando expressões nesta ordem, se existirem.

A chamada de um procedimento é isolada de outros comandos. Não poderá substituir valor em expressões ou ser associada à entrada ou saída diretamente.

Programa em SCILAB:

```
// Exemplo 8.
// Calcular e mostrar o valor do numero (e) com varias aproximacoes.
//
// definir dado global (e)
  global E;              // valor de (e) com (x) termos
// definir procedimento com parametro
function EXP1 ( N )
//
  global E;
//
// 2.1 definir dados locais
  X = 0;                 // contador
  PRODUTO = 1;     // produto de naturais
  SOMA       = 1.0;   // soma de inversos
// 2.2. gerar (N-1) numeros naturais
  for ( X = 1 : 1 : (N-1) )
  // calcular o fatorial
    PRODUTO = PRODUTO * X;
  // acumular a soma dos inversos
    SOMA = SOMA + 1.0/PRODUTO;
  end // fim da geracao de (N-1) naturais
// 2.3 definir valor de retorno na variavel global
  E = SOMA;
endfunction // procedimento EXP1( )
//
// parte principal
//
// calcular e mostrar o valor do numero (e) com varias aproximacoes
// 3. mostrar resultado
  clc;   // limpar a area de trabalho
  EXP1 ( 10 );
  printf ( "\ne com 10 termos = %f", E );
  EXP1 ( 20 );
  printf ( "\ne com 20 termos = %f", E );
// pausa para terminar
  printf ( "\nPressionar ENTER para terminar.\n" );
  halt;
// fim do programa
```

Programa em C:

```
// Exemplo 8.
// Calcular e mostrar o valor do numero (e) com varias aproximacoes.
//
// bibliotecas necessarias
#include <stdio.h>
#include <stdlib.h>
//
// definir dado global (e)
   float E;                          // valor de (e) com (x) termos
//
// definir procedimento com parametro
void EXP1 ( int N )
{
 // 2.1 definir dados locais
   int    X,                         // contador
          PRODUTO = 1;               // produto de naturais
   float SOMA = 1.0,                 // soma de inversos
// 2.2. gerar (N-1) numeros naturais
   for ( X = 1; X <= (N-1); X = X +1 )
   {
     // calcular o fatorial
     PRODUTO = PRODUTO * X;
     // acumular a soma dos inversos
    SOMA = SOMA + 1.0/PRODUTO;
   } // fim da geracao de (N-1) naturais
 // 2.3 definir valor de retorno na variavel global
    E = SOMA;
} // fim do procedimento EXP1( )
//
// parte principal
//
int main (void)
{
// 3. mostrar resultado
   EXP1 ( 10 );
   printf ( “\ne com 10 termos = %f“, E );
   EXP1 ( 20 );
   printf ( “\ne com 20 termos = %f“, E );
// pausa para terminar
   printf ( "Pressionar ENTER para terminar." );
   getchar ( );
   return ( 0 );
} // fim do programa
```

Programa em C++:

```
// Exemplo 8.
// Calcular e mostrar o valor do numero (e) com varias aproximacoes.
//
// bibliotecas necessarias
#include <iostream>
using namespace std;
//
// definir dado global (e)
   double E;                    // valor de (e) com (x) termos
//
// definir procedimento com parametro
void EXP1 ( int N )
{
 // 2.1 definir dados locais
   int       X,                 // contador
             PRODUTO = 1;       // produto de naturais
   double SOMA = 1.0,           // soma de inversos
// 2.2. gerar (N-1) numeros naturais
   for ( X = 1; X <= (N-1); X = X +1 )
   {
     // calcular o fatorial
     PRODUTO = PRODUTO * X;
     // acumular a soma dos inversos
    SOMA = SOMA + 1.0/PRODUTO;
   } // fim da geracao de (N-1) naturais
 // 2.3 definir valor de retorno na variavel global
    E = SOMA;
} // fim do procedimento EXP1( )
//
// parte principal
//
int main (void)
{
// 3. mostrar resultado
   EXP1 ( 10 );
   cout << "\ne com 10 termos = " << E;
   EXP1 ( 20 );
   cout << "\ne com 20 termos = " << E;
// pausa para terminar
   cout << "Pressionar ENTER para terminar.";
   cin.get ( );
   return EXIT_SUCCESS;
} // fim do programa
```

Programa em C#:

```
/*
 * Exemplo 8
 * Calcular e mostrar o valor do numero (e) com varias aproximacoes.
 */
using System;

class Exemplo_8
{
// 1. definir dado global (e)
   public static double E;      // valor de (e) com (x) termos
//
// 2. definir procedimento com parametro
   public static void EXP1 ( int N )
   {
   // 2.1. definir dados locais
      int X,               // contador
          PRODUTO = 1;         // produto de naturais
      double SOMA = 1.0;      // soma de inversos
   // 2.2. gerar (N-1) numeros naturais
      for ( X = 1; X <= (N-1); X = X+1 )
      {
      // 2.2.1. calcular o fatorial
         PRODUTO = PRODUTO * X;
      // 2.2.2. acumular a soma dos inversos
         SOMA = SOMA + 1.0 / PRODUTO;
      } // fim da geracao de (N-1) naturais
   // 2.3 definir valor de retorno na variavel global
      E = SOMA;
   } // fim do procedimento EXP1 ( )
//
// parte principal
//
   public static void Main ( )
   {
   // 3. mostrar resultado
      EXP1 ( 10 );
      Console.WriteLine ( "\ne com 10 termos = " + E );
      EXP1 ( 20 );
      Console.WriteLine ( "\ne com 20 termos = " + E );
   // pausa para terminar
      Console.Write ( "\nPressionar ENTER para terminar." );
      Console.ReadLine ( );
   } // end Main ( )

} // fim Exemplo_8 class
```

Programa em Java:

```
/**
  * Exemplo 8
  * Calcular e mostrar o valor do numero (e) com varias aproximacoes.
  */

// ------------------------------------------------ classes necessarias

// ------------------------------------------------ definicao de classe

class Exemplo_8
{
// 1. definir dado global (e)
   public static double E;      // valor de (e) com (x) termos
//
// 2. definir procedimento com parametro
   public static void EXP1 ( int N )
   {
   // 2.1. definir dados locais
      int X,                    // contador
          PRODUTO = 1;          // produto de naturais
      double SOMA = 1.0;        // soma de inversos
   // 2.2. gerar (N-1) numeros naturais
      for ( X = 1; X <= (N-1); X = X+1 )
      {
      // 2.2.1. calcular o fatorial
         PRODUTO = PRODUTO * X;
      // 2.2.2. acumular a soma dos inversos
         SOMA = SOMA + 1.0 / PRODUTO;
      } // fim da geracao de (N-1) naturais
   // 2.3 definir valor de retorno na variavel global
      E = SOMA;
   } // fim do procedimento EXP1 ( )
//
// parte principal
//
   public static void main ( String [ ] args )
   {
   // 3. mostrar resultado
      EXP1 ( 10 );
      System.out.println ( "\ne com 10 termos = " + E );
      EXP1 ( 20 );
      System.out.println ( "\ne com 20 termos = " + E );
   // pausa para terminar
      System.out.print ( "\nPressionar ENTER para terminar." );
      System.console( ).readLine( );
   } // end main ( )

} // fim Exemplo_8 class
```

Programa em Python:

```
# Exemplo 8.
# Calcular e mostrar o valor do numero (e) com varias aproximacoes.
#
# definir dado global (e)
E = 0;    # valor de (e) com (x) termos
# definir procedimento com parametro
def EXP1 ( N ):
#
   global E;
#
# 2.1 definir dados locais
   X = 0;                    # contador
   PRODUTO = 1;              # produto de naturais
   SOMA    = 1.0; # soma de inversos
# 2.2. gerar (N-1) numeros naturais
   for X in range ( 1, (N-1)+1, 1 ):
   # calcular o fatorial
      PRODUTO = PRODUTO * X;
   # acumular a soma dos inversos
      SOMA = SOMA + 1.0/PRODUTO;
  # fim da geracao de (N-1) naturais
# 2.3 definir valor de retorno na variavel global
   E = SOMA;
# procedimento EXP1( )
#
# parte principal
#
# calcular e mostrar o valor do numero (e) com varias aproximacoes
# 3. mostrar resultado
EXP1 ( 10 );
print   ( "\ne com 10 termos = ", E );
EXP1 ( 20 );
print   ( "\ne com 20 termos = ", E );
# pausa para terminar
print   ( "\nPressionar ENTER para terminar.\n" );
input  ( );
# fim do programa
```

Exercícios

1. Escrever um procedimento para calcular o somatório abaixo :

$$S(n) = \frac{1}{0!} + \frac{1}{1!} + \frac{1}{2!} + \frac{1}{3!} + \quad \cdots + \frac{1}{n!}$$

recebendo o número de termos (N) como parâmetro.

2. Escrever um procedimento para calcular com 20 parcelas :

$$e^x = \frac{x^0}{0!} + \frac{x^1}{1!} + \frac{x^2}{2!} + \frac{x^3}{3!} + \quad \cdots$$

recebendo um valor real (X) como parâmetro.

3. Escrever um procedimento para gerar o cosseno de um ângulo (X), em radianos,

$$\cos(x,n) = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \quad \cdots$$

recebendo um valor real (X) como parâmetro, e
para (N) termos, com (N) lido do teclado.

4. Escrever um procedimento para gerar o seno de um ângulo (X), em radianos,

$$sen(x,n) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \quad \cdots$$

recebendo um valor real (X) como parâmetro, e
para (N) termos, com (N) lido do teclado.

5. Escrever um procedimento para calcular o somatório :

$$S(x) = \frac{x^{50}}{0!} - \frac{x^{49}}{1!} + \frac{x^{48}}{2!} - \quad \cdots \quad - \frac{x^1}{49!} + \frac{x^0}{50!}$$

recebendo um valor real (X) como parâmetro.

- Procedimentos e funções

Procedimentos podem ser usados em combinação com funções. A regra básica é fazer a definição primeiro antes de usá-la, não importando o tipo do método. Se não houver qualquer comprometimento, em geral, as funções poderão ser definidas primeiro, antes dos procedimentos.

Exemplo 9:

Calcular e mostrar o valor do número (e), com aproximações de 10 e 20 termos da soma, usando uma função para calcular o fatorial:

$$e = 1 + \frac{1}{1!} + \frac{1}{2!} + \frac{1}{3!} + \quad \cdots$$

Análise de dados:

- Dados da função para calcular o fatorial:

Dado	Tipo	Valor Inicial	Obs.
N	inteiro		número de termos (parâmetro)
X	inteiro		contador
PRODUTO	inteiro	1	produto dos termos

- Fórmulas que relacionam os dados :

PRODUTO = PRODUTO * X; para cada número natural menor ou igual a N

- Dados do procedimento:

Dado	Tipo	Valor Inicial	Obs.
N	inteiro		número de termos (parâmetro)
X	inteiro		contador
SOMA	real	1.0	somatório de (N) termos

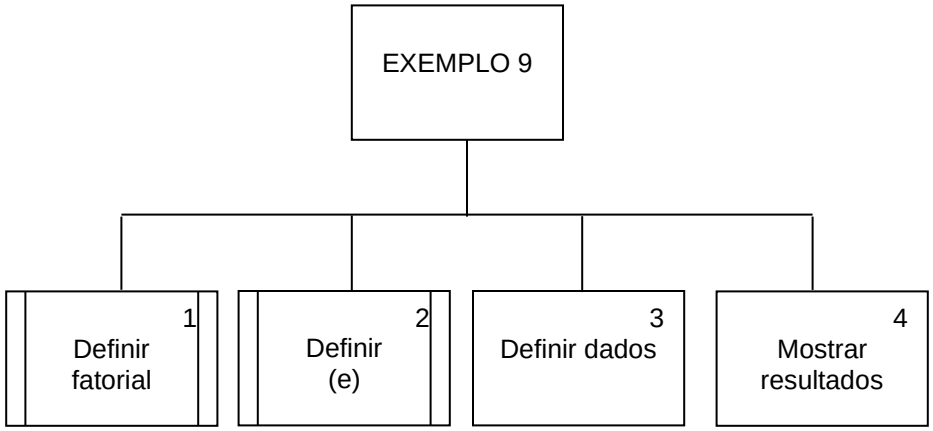
- Fórmulas que relacionam os dados :

SOMA = SOMA + 1/fatorial(I); para cada produto calculado

- Dados da parte principal:

Dado	Tipo	Valor Inicial	Obs.
e	real		valor do número (e)

Diagrama funcional:



Algoritmo, com procedimento:

Esboço:

Primeira versão, só comentários.

Exemplo 9	v.1
Ação	Bloco
! definir função FATORIAL (inteiro N)	1
! definir procedimento EXP (inteiro N)	2
! definir dados da parte principal	3
! mostrar resultado	4

Segunda versão, só comentários.
Não haverá dados na parte principal, mas será necessário uma variável global (e).

Exemplo 9	v.2
Ação	Bloco
! definir dado global (e)	1
! definir função FATORIAL (inteiro N)	2
! definir procedimento EXP1 (inteiro N)	3
! parte principal	
! mostrar resultado	4

Terceira versão, refinar o terceiro e quarto blocos.

Exemplo 9	v.3
Ação	Bloco
! definir dado global (e)	1
! definir função	2
inteiro FATORIAL (inteiro N) ! com parâmetros	
! definição de dados inteiro I, ! contador PRODUTO = 1; ! produto de naturais	2.1
! gerar números naturais em ordem decrescente	2.2
repetir para (X = N : -1 : 1) ! multiplicar cada número natural PRODUTO = PRODUTO * X; fim repetir ! da geração de N naturais	2.2.1
retornar (PRODUTO);	
! definir procedimento EXP1 (inteiro N)	3
procedimento EXP1 (inteiro N)	
! definição de dados locais inteiro X; ! contador real SOMA = 1.0, ! soma de inversos	3.1
! gerar (N-1) naturais em ordem crescente	3.2
repetir (X = 1 : 1 : (N-1)) ! acumular a soma dos inversos SOMA = SOMA + 1.0/FATORIAL(X); fim repetir ! da geração de (N-1) naturais	
! definição do valor de retorno E = SOMA;	3.3
! parte principal	
! mostrar resultado EXP1(10); tela ← (“\ne com 10 termos = “, E); EXP1(20); tela ← (“\ne com 20 termos = “, E);	4

Programa em SCILAB:

```
// Exemplo 9.
// Calcular e mostrar o valor do numero (e) com varias aproximacoes.
//
// definir dado global (e)
  global E;              // valor de (e) com (x) termos
//
// 2. definir funcao FATORIAL com parametro
function retorno = FATORIAL ( N )
// 2.1. definir dados
  X = 0;                 // contador
  PRODUTO = 1;     // produto de naturais
// 2.2. gerar N numeros naturais
  for ( X = N : (-1) : 1 )
  // 2.2.1 multiplicar cada numero natural
    PRODUTO = PRODUTO * X;
  end // fim do bloco repetitivo
  // retorno do valor calculado
  retorno =  PRODUTO;
endfunction // FATORIAL( )
//
// definir procedimento com parametro
function EXP01 ( N )
//
  global E;
//
// 2.1 definir dados locais
   X = 0;                // contador
   SOMA = 1.0;       // soma de inversos
// 2.2. gerar N numeros naturais
  for ( X = 1 : 1 : (N-1) )
  // acumular a soma dos inversos
    SOMA = SOMA + 1.0/FATORIAL ( X );
  end // fim da geracao de (N-1) naturais
// 2.3 definir valor de retorno na variavel global
  E = SOMA;
endfunction // procedimento EXP1( )
//
// parte principal
//
// calcular e mostrar o valor do numero (e) com varias aproximacoes
// 3. mostrar resultado
  clc;   // limpar a area de trabalho
  EXP01 ( 10 );
  printf ( "\ne com 10 termos = %f", E );
  EXP01 ( 20 );
  printf ( "\ne com 20 termos = %f", E );
// pausa para terminar
  printf ( "\nPressionar ENTER para terminar.\n" );
  halt;
// fim do programa
```

Programa em C:

```
// Exemplo 9.
// Calcular e mostrar o valor do numero (e) com varias aproximacoes.
//
// bibliotecas necessarias
#include <stdio.h>
#include <stdlib.h>
//
// 1. definir dado global (e)
   float E;                        // valor de (e) com (x) termos
//
// 2. definir funcao FATORIAL com parametro
int FATORIAL ( int N )
{
 // 2.1. definir dados
   int X,                          // contador
       PRODUTO = 1;                // produto de naturais
 // 2.2. gerar N numeros naturais
   for ( X = N; X >= 1; X = X - 1)
   { // 2.2.1 multiplicar cada numero natural
       PRODUTO = PRODUTO * X;
   } // fim do bloco repetitivo
   // retorno do valor calculado
   return ( PRODUTO );
} // fim FATORIAL( )

// 3. definir do procedimento EXP1 com parametro
void EXP1 ( int N )
{
 // 3.1 definir dados locais
   int    X;                       // contador
   float SOMA = 1.0,               // soma de inversos
 // 3.2. gerar N numeros naturais
   for ( X = 1; X <= (N-1); X = X +1 )
   { // acumular a soma dos inversos
       SOMA = SOMA + 1.0/FATORIAL( X );
   } // fim da geracao de (N-1) naturais
 // 3.3 definir valor de retorno na variavel global
    E = SOMA;
} // fim do procedimento EXP1
//
// parte principal
//
int main (void)
{
// 4. mostrar resultado
   EXP1 ( 10 );
   printf ( “\ne com 10 termos = %f“, E );
   EXP1 ( 20 );
   cout << “\ne com 20 termos = “ << E;
// pausa para terminar
   printf ( "Pressionar ENTER para terminar." );
   getchar ( );
   return ( 0 );
} // fim do programa
```

Programa em C++:

```
// Exemplo 9.
// Calcular e mostrar o valor do numero (e) com varias aproximacoes.
//
// bibliotecas necessarias
#include <iostream>
using namespace std;
//
// 1. definir dado global (e)
   double E;                    // valor de (e) com (x) termos
//
// 2. definir funcao FATORIAL com parametro
int FATORIAL ( int N )
{
 // 2.1. definir dados
   int X,                       // contador
       PRODUTO = 1;             // produto de naturais
 // 2.2. gerar N numeros naturais
   for ( X = N; X >= 1; X = X - 1)
   { // 2.2.1 multiplicar cada numero natural
       PRODUTO = PRODUTO * X;
   } // fim do bloco repetitivo
   // retorno do valor calculado
   return ( PRODUTO );
} // fim FATORIAL( )

// 3. definir do procedimento EXP1 com parametro
void EXP1 ( int N )
{
 // 3.1 definir dados locais
   int       X;                 // contador
   double SOMA = 1.0,           // soma de inversos
 // 3.2. gerar N numeros naturais
   for ( X = 1; X <= (N-1); X = X +1 )
   { // acumular a soma dos inversos
       SOMA = SOMA + 1.0/FATORIAL( X );
   } // fim da geracao de (N-1) naturais
 // 3.3 definir valor de retorno na variavel global
    E = SOMA;
} // fim do procedimento EXP1
//
// parte principal
//
int main (void)
{
// 4. mostrar resultado
   EXP1 ( 10 );
   cout << “\ne com 10 termos = “ << E;
   EXP1 ( 20 );
   cout << “\ne com 20 termos = “ << E;
// pausa para terminar
   cout << "Pressionar ENTER para terminar.";
   cin.get ( );
   return EXIT_SUCCESS;
} // fim do programa
```

Programa em C#:

```
/**
  * Exemplo 9
  * Calcular e mostrar o valor do numero (e) com varias aproximacoes.
  */
using System;

class Exemplo_9
{
// 1. definir dado global (e)
   public static double E;      // valor de (e) com (x) termos
// 2. definir funcao FATORIAL com parametro
   public static int FATORIAL ( int N )
   { // 2.1. definir dados
        int X,                  // contador
            PRODUTO = 1;        // produto de naturais
     // 2.2. gerar N numeros naturais
        for ( X = N; X >= 1; X = X-1 )
        {
        // 2.2.1. multiplicar cada numero natural
          PRODUTO = PRODUTO * X;
        } // fim do bloco repetitivo
   // retorno do valor calculado
     return ( PRODUTO );
   } // fim da funcao FATORIAL ( )
// 3. definir procedimento com parametro
   public static void EXP1 ( int N )
   { // 3.1. definir dados locais
     int X;                     // contador
     double SOMA = 1.0;         // soma de inversos
   // 3.2. gerar N numeros naturais
     for ( X = 1; X <= (N-1); X = X+1 )
     { // 3.2.1. acumular a soma dos inversos
         SOMA = SOMA + 1.0 / FATORIAL ( X );
     } // fim da geracao de (N-1) naturais
   // 3.3 definir valor de retorno na variavel global
     E = SOMA;
   } // fim do procedimento EXP1 ( )
//
// parte principal
//
   public static void Main ( )
   {
   // 4. mostrar resultado
     EXP1 ( 10 );               Console.WriteLine ( "\ne com 10 termos = " + E );
     EXP1 ( 20 );               Console.WriteLine ( "\ne com 20 termos = " + E );
   // pausa para terminar
     Console.Write ( "\nPressionar ENTER para terminar." );
     Console.ReadLine ( );
   } // end Main ( )

} // fim Exemplo_9 class
```

Programa em Java:

```
/**
  * Exemplo 9
  * Calcular e mostrar o valor do numero (e) com varias aproximacoes.
  */

// ------------------------------------------------ classes necessarias

// ------------------------------------------------ definicao de classe

class Exemplo_9
{
// 1. definir dado global (e)
   public static double E;      // valor de (e) com (x) termos
// 2. definir funcao FATORIAL com parametro
   public static int FATORIAL ( int N )
   { // 2.1. definir dados
        int X,                  // contador
            PRODUTO = 1;        // produto de naturais
     // 2.2. gerar N numeros naturais
        for ( X = N; X >= 1; X = X-1 )
        {
        // 2.2.1. multiplicar cada numero natural
           PRODUTO = PRODUTO * X;
        } // fim do bloco repetitivo
   // retorno do valor calculado
      return ( PRODUTO );
   } // fim da funcao FATORIAL ( )
// 3. definir procedimento com parametro
   public static void EXP1 ( int N )
   { // 3.1. definir dados locais
      int X;                // contador
      double SOMA = 1.0;    // soma de inversos
   // 3.2. gerar N numeros naturais
      for ( X = 1; X <= (N-1); X = X+1 )
      { // 3.2.1. acumular a soma dos inversos
          SOMA = SOMA + 1.0 / FATORIAL ( X );
      } // fim da geracao de (N-1) naturais
   // 3.3 definir valor de retorno na variavel global
      E = SOMA;
   } // fim do procedimento EXP1 ( )
//
// parte principal
//
   public static void main ( String [ ] args )
   {
   // 4. mostrar resultado
      EXP1 ( 10 );              System.out.println ( "\ne com 10 termos = " + E );
      EXP1 ( 20 );              System.out.println ( "\ne com 20 termos = " + E );
   // pausa para terminar
      System.out.print ( "\nPressionar ENTER para terminar." );
      System.console( ).readLine( );
   } // end main ( )
} // fim Exemplo_9 class
```

Programa em Python:

```
# Exemplo 9.
# Calcular e mostrar o valor do numero (e) com varias aproximacoes.
#
# definir dado global (e)
E = 0;              # valor de (e) com (x) termos
#
# 2. definir funcao FATORIAL com parametro
def FATORIAL ( N ):
# 2.1. definir dados
  X = 0;                    # contador
  PRODUTO = 1;              # produto de naturais
# 2.2. gerar N numeros naturais
  for X in range ( N, 1-1, -1 ):
  # 2.2.1 multiplicar cada numero natural
    PRODUTO = PRODUTO * X;
  # fim do bloco repetitivo
# retorno do valor calculado
  return ( PRODUTO );
# FATORIAL( )
#
# definir procedimento com parametro
def EXP01 ( N ):
#
  global E;
#
# 2.1 definir dados locais
  X    = 0;          # contador
  SOMA = 1.0;  # soma de inversos
# 2.2. gerar N numeros naturais
  for X in range (  1, (N-1)+1, 1 ):
  # acumular a soma dos inversos
    SOMA = SOMA + 1.0/FATORIAL ( X );
  # fim da geracao de (N-1) naturais
# 2.3 definir valor de retorno na variavel global
  E = SOMA;
# procedimento EXP1( )
#
# parte principal
#
# calcular e mostrar o valor do numero (e) com varias aproximacoes
# 3. mostrar resultado
EXP01 ( 10 );
print     ( "\ne com 10 termos = ", E );
EXP01 ( 20 );
print     ( "\ne com 20 termos = ", E );
# pausa para terminar
print     ( "\nPressionar ENTER para terminar.\n" );
input    ( );
# fim do programa
```

Exercícios

1. Escrever uma função para calcular o fatorial e outra função para calcular o somatório abaixo :

$$S(n) = \frac{1}{0!} + \frac{1}{1!} + \frac{1}{2!} + \frac{1}{3!} + \quad \cdots + \frac{1}{n!}$$

recebendo o número de termos (N) como parâmetro.

2. Escrever um procedimento para calcular o fatorial e uma função para calcular com 20 parcelas :

$$e^x = \frac{x^0}{0!} + \frac{x^1}{1!} + \frac{x^2}{2!} + \frac{x^3}{3!} + \quad \cdots$$

recebendo um valor real (X) como parâmetro.

3. Escrever um procedimento para calcular o fatorial e um procedimento para calcular o cosseno de um ângulo (X), em radianos,

$$\cos(x,n) = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \quad \cdots$$

recebendo um valor real (X) como parâmetro, e
para (N) termos, com (N) lido do teclado.

4. Escrever um procedimento para calcular o fatorial e uma função para gerar o seno de um ângulo (X), em radianos,

$$sen(x,n) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \quad \cdots$$

recebendo um valor real (X) como parâmetro, e
para (N) termos, com (N) lido do teclado.

5. Escrever uma função para a exponenciação, outra para o fatorial e um procedimento para calcular o somatório :

$$S(x) = \frac{x^{50}}{0!} - \frac{x^{49}}{1!} + \frac{x^{48}}{2!} - \quad \cdots \quad - \frac{x^1}{49!} + \frac{x^0}{50!}$$

recebendo um valor real (X) como parâmetro.

Sugestão: Para fazer a exponenciação considerar as operações abaixo.

$x^y = z = \exp(y * \ln (x))$

Mecanismo de passagem de parâmetros

Quando passados aos procedimentos e às funções, os parâmetros podem ser qualificados pela ação de importação, ou exportação.

Em um procedimento ou função, considera-se a importação como uma cópia do valor do parâmetro atual (ou real), no momento da chamada, para o valor do parâmetro formal correspondente na definição. O parâmetro formal, dentro do escopo do procedimento, pode ser alterado, mantendo intacto o valor externo, se a passagem foi feita apenas por valor.

Se for desejado efetuar uma alteração em ambos, tanto no parâmetro formal quanto no atual, o nome do parâmetro formal deverá ser precedido por um qualificativo ("&"), significando uma referência imediata (endereço) da variável passada como parâmetro atual.

Esses elementos qualificados como exportáveis podem alterar o valor das variáveis passadas como parâmetros. Se estes elementos forem alterados dentro do procedimento, os valores das variáveis externas também o serão. Os parâmetros formais exportáveis devem ser, sempre, variáveis; não podem receber constantes, ou expressões, no momento da chamada. Os parâmetros marcados como exportáveis também podem levar valores para os procedimentos (ou funções), e retornar com valores diferentes daqueles iniciais.

As funções, preferencialmente, devem receber apenas parâmetros importados (passados apenas por valor), e exportar (retornar) um único valor do tipo que define a função, em seu próprio nome.

Não são impostos critérios sobre quando utilizar uma, ou outra, forma de passagem, mas de modo geral, a **passagem por valor** deverá ser preferida, pois mantém os valores do contexto da chamada. Estes valores poderão ser modificados pela **passagem por referência**. O uso de variáveis globais, entretanto, deverá ser evitado, na medida do possível.

Exemplo:

Fazer um procedimento genérico para ler um valor inteiro do teclado.

```
procedimento Ler_Inteiro ( inteiro & X )
  tela ← ("\nFornecer um valor inteiro: "); X ← teclado;
fim procedimento ! Ler_Inteiro( )
```

Exemplo:

Fazer um procedimento genérico para ler um valor positivo do teclado.

```
procedimento Ler_Positivo ( int & X )
  repetir até ( X >= 0 )
    tela ← ("\nFornecer um valor positivo: "); X ← teclado;
  fim repetir ! enquanto ( X < 0 )
fim procedimento ! Ler_Positivo( )
```

Exercícios

1. Escrever um procedimento com passagem por referência para ler do teclado um valor inteiro, mas que também possa exibir uma mensagem passada como parâmetro.

2. Escrever um procedimento com passagem por referência para ler do teclado um valor inteiro positivo, mas que também possa exibir uma mensagem passada como parâmetro.

3. Escrever um procedimento com passagem por referência para ler do teclado um valor inteiro não nulo, mas que também possa exibir uma mensagem passada como parâmetro.

4. Escrever um procedimento com passagem por referência para ler do teclado um valor inteiro maior que outro, passado como parâmetro, e também possa exibir uma mensagem passada como parâmetro.

5. Escrever um procedimento com passagem por referência para ler do teclado um caractere como resposta a uma pergunta como parâmetro, que aceite apenas (S ou N) como valores válidos.

Exemplo 10:

Calcular e mostrar o valor do número (e), com aproximações de 10 e 20 termos da soma, usando uma função para calcular o fatorial e um procedimento com passagem por referência:

$$e = 1 + \frac{1}{1!} + \frac{1}{2!} + \frac{1}{3!} + \quad \cdots$$

Análise de dados:

- Dados da função para calcular o fatorial:

Dado	Tipo	Valor Inicial	Obs.
N	inteiro		número de termos (parâmetro)
X	inteiro		contador
PRODUTO	inteiro	1	produto dos termos

- Fórmulas que relacionam os dados :

PRODUTO = PRODUTO * I; para cada número natural menor ou igual a N

- Dados do procedimento:

Dado	Tipo	Valor Inicial	Obs.
N	inteiro		número de termos
			(parâmetro com passagem por valor)
SOMA	real	1.0	somatório de (N) termos
			(parâmetro com passagem por referência)
X	inteiro		contador

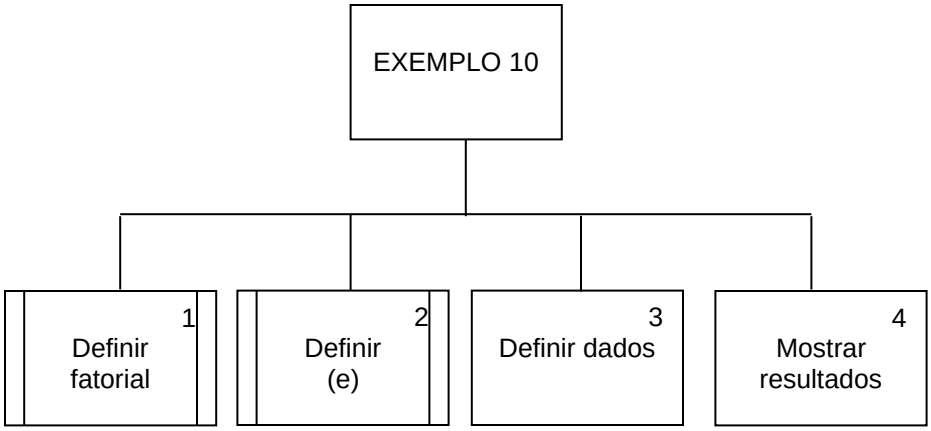
- Fórmulas que relacionam os dados :

SOMA = SOMA + 1/fatorial(I); para cada produto calculado

- Dados da parte principal:

Dado	Tipo	Valor Inicial	Obs.
E	real		valor do número (e)

Diagrama funcional:



Algoritmo, com procedimento:

Esboço:

Primeira versão, só comentários.

Exemplo 10	v.1
Ação	Bloco
! definir função FATORIAL (inteiro N)	1
! definir procedimento EXP (inteiro N)	2
! definir dados da parte principal	3
! mostrar resultado	4

Segunda versão, só comentários.
Não será mais necessário a definição de uma variável global (e).

Exemplo 10	v.2
Ação	Bloco
! definir função FATORIAL (inteiro N)	1
! definir procedimento EXP1 (inteiro N, real & e)	2
! definir dados da parte principal	3
! mostrar resultado	4

Terceira versão, refinamento do terceiro bloco.

Exemplo 10	v.3
Ação	Bloco
! definir função FATORIAL (inteiro N)	1
! definir procedimento EXP1 (inteiro N, real & e)	2
! definir dados da parte principal	3
! mostrar resultado	4

Quarta versão, refinar o terceiro e quarto blocos.

Exemplo 10	v.4
Ação	Bloco
! definir função	1
inteiro FATORIAL (inteiro N) ! com parâmetros	
! definir dados locais inteiro X, ! contador PRODUTO = 1; ! produto de naturais	1.1
! gerar números naturais em ordem decrescente	1.2
repetir para (X = N : -1 : 1) ! multiplicar cada número natural PRODUTO = PRODUTO * X; fim repetir ! a geração de N naturais	1.2.1
return (PRODUTO);	
! definir procedimento EXP1 (inteiro N, real SOMA)	2
procedimento EXP1 (inteiro N, real & SOMA)	
! definir dados locais inteiro X; ! contador SOMA=1.0, ! valor inicial da soma	2.1
! gerar (N-1) naturais em ordem crescente	2.2
repetir para (X = 1 : 1 : (N-1)) ! acumular a soma dos inversos SOMA = SOMA + 1.0/FATORIAL(X); fim repetir ! da geração de (N-1) naturais	2.3
! parte principal	
! definir dados da parte principal real E; ! valor calculado do número (e)	3
! mostrar resultado EXP(10, E); tela ← (“\ne com 10 termos = “, E); EXP(20, E); tela ← (“\ne com 20 termos = “, E);	4

Programa em SCILAB:

```
// Exemplo 10.
// Calcular e mostrar o valor do numero (e) com varias aproximacoes.
//
// 2. definir a funcao FATORIAL com parametro
function retorno = FATORIAL ( N )
// 2.1. definir dados
  X         = 0;          // contador
  PRODUTO = 1;            // produto de naturais
// 2.2. gerar N numeros naturais
  for ( X = N : (-1) : 1 )
  // 2.2.1 multiplicar cada numero natural
    PRODUTO = PRODUTO * X;
  end // fim do bloco repetitivo
  // retorno do valor calculado
  retorno =  PRODUTO;
endfunction // FATORIAL( )
//
// definir procedimento com parametro
function EXP01 ( N, SOMA )
//
// 2.1 definir dados locais
// int X;                  // contador
  SOMA = 1.0;             // soma de inversos
// 2.2. gerar (N-1) numeros naturais
  for ( X = 1 : 1 : (N-1) )
  // acumular a soma dos inversos
    SOMA = SOMA + 1.0/FATORIAL ( X );
  end // fim da geracao de (N-1) naturais
endfunction // procedimento EXP01( )
//
// parte principal
//
// calcular e mostrar o valor do numero (e) com varias aproximacoes
// definir dado
   E = 0.0;               // valor de (e) com (x) termos
// 3. mostrar resultado
  clc;  // limpar a area de trabalho
  EXP01 ( 10, E );
  printf ( "\ne com 10 termos = %f", E );
  EXP01 ( 20, E );
  printf ( "\ne com 20 termos = %f", E );
// pausa para terminar
  printf ( "\nPressionar ENTER para terminar.\n" );
  halt;
// fim do programa
```

Programa em C:

```
// Exemplo 10.
// Calcular e mostrar o valor do numero (e) com varias aproximacoes.
//
// bibliotecas necessarias
#include <stdio.h>
#include <stdlib.h>
//
// 1. definir funcao FATORIAL com parametro passado por valor
int FATORIAL ( int N )
{
 // 1.1. definir dados
   int X,                          // contador
       PRODUTO = 1;                // produto de naturais
 // 1.2. gerar N numeros naturais
   for ( X = N; X >= 1; X = X – 1 )
   { // 2.2.1 multiplicar cada número natural
     PRODUTO = PRODUTO * X;
   } // fim do bloco repetitivo
   // retorno do valor calculado
   return ( PRODUTO );
} // fim FATORIAL( )

// 2. definir procedimento EXP1 com parametros passados por valor e referencia
void EXP1 ( int N, float * SOMA )
{
 // 2.1 definir dados locais
   int    X;                       // contador
   *SOMA = 1.0,                    // valor inicial da soma de inversos
 // 2.2. gerar (N-1) numeros naturais
   for ( I = 1; I <= (N-1); I = I +1 )
   { // acumular a soma dos inversos
      *SOMA = *SOMA + 1.0/FATORIAL( I );
   } // fim da geracao de (N-1) naturais
} // fim do procedimento EXP1
//
// parte principal
//
int main (void)
{
// 3. definir dados
   float E;                        // valor de (e) com (x) termos
//
// 4. mostrar resultado
   EXP ( 10, &E );
   printf ( “\ne com 10 termos = %f“, E );
   EXP ( 20, &E );
   printf ( “\ne com 20 termos = %f“, E );
// pausa para terminar
   printf ( "Pressionar ENTER para terminar." );
   getchar ( );
   return ( 0 );
} // fim do programa
```

Programa em C++:

```
// Exemplo 10.
// Calcular e mostrar o valor do numero (e) com varias aproximacoes.
//
// bibliotecas necessarias
#include <iostream>
using namespace std;
//
// 1. definir funcao FATORIAL com parametro passado por valor
int FATORIAL ( int N )
{
 // 1.1. definir dados
   int X,                          // contador
       PRODUTO = 1;          // produto de naturais
 // 1.2. gerar N numeros naturais
   for ( X = N; X >= 1; X = X – 1 )
   { // 2.2.1 multiplicar cada número natural
     PRODUTO = PRODUTO * X;
   } // fim do bloco repetitivo
   // retorno do valor calculado
   return ( PRODUTO );
} // fim FATORIAL( )

// 2. definir procedimento EXP1 com parametros passados por valor e referencia
void EXP1 ( int N, double & SOMA )
{
 // 2.1 definir dados locais
   int    X;                       // contador
   SOMA = 1.0,                  // valor inicial da soma de inversos
 // 2.2. gerar (N-1) numeros naturais
   for ( I = 1; I <= (N-1); I = I +1 )
   { // acumular a soma dos inversos
    SOMA = SOMA + 1.0/FATORIAL( I );
   } // fim da geracao de (N-1) naturais
} // fim do procedimento EXP1
//
// parte principal
//
int main (void)
{
// 3. definir dados
   double E;                      // valor de (e) com (x) termos
//
// 4. mostrar resultado
   EXP ( 10, E );
   cout << “\ne com 10 termos = “ << E;
   EXP ( 20, E );
   cout << “\ne com 20 termos = “ << E;
// pausa para terminar
   cout << "Pressionar ENTER para terminar.";
   cin.get ( );
   return EXIT_SUCCESS;
} // fim do programa
```

Programa em C#:

```
/*
 * Exemplo 10a
 * Calcular e mostrar o valor do numero (e) com varias aproximacoes.
 */
using System;

class Exemplo_10
{
//
// 2. definir funcao FATORIAL com parametro
   public static int FATORIAL ( int N )
   {
   // 2.1. definir dados
      int X,                        // contador
          PRODUTO = 1;         // produto de naturais
   // 2.2. gerar N numeros naturais
      for ( X = N; X >= 1; X = X-1 )
      {
      // 2.2.1. multiplicar cada numero natural
         PRODUTO = PRODUTO * X;
      } // fim do bloco repetitivo
   // retorno do valor calculado
      return ( PRODUTO );
   } // fim da funcao FATORIAL ( )

//
// 3. definir procedimento com parametro
   public static void EXP1 ( int N, ref double E )
   {
   // 3.1. definir dados locais
      int X;                 // contador
      double SOMA = 1.0;       // soma de inversos
   // 3.2. gerar (N-1) numeros naturais
      for ( X = 1; X <= (N-1); X = X+1 )
      {
      // 3.2.1. acumular a soma dos inversos
         SOMA = SOMA + 1.0 / FATORIAL ( X );
      } // fim da geracao de (N-1) naturais
   // 3.3 definir valor de retorno na variavel global
      E = SOMA;
   } // fim do procedimento EXP1 ( )
//
```

```
// parte principal
//
   public static void Main ( )
   {
   // 3. definir dados
      double E = 0.0;  // valor de (e) com (x) termos
   // 4. mostrar resultado
      EXP1 ( 10, ref E );
      Console.WriteLine ( "\ne com 10 termos = " + E.valor );
      EXP1 ( 20, E );
      Console.WriteLine ( "\ne com 20 termos = " + E.valor );
   // pausa para terminar
      Console.Write ( "\nPressionar ENTER para terminar." );
      Console.ReadLine ( );
   } // end Main ( )

} // fim Exemplo_10 class
```

Programa em Java:

```
/**
  * Exemplo 10a
  * Calcular e mostrar o valor do numero (e) com varias aproximacoes.
  */

// ------------------------------------------------ classes necessarias

// ------------------------------------------------ definicao de classes

// definir um envelope para o objeto
class e_Wrapper
{
// definir o objeto armazenador do dado
   public double valor;

// definir como alterar o valor do objeto
   public e_Wrapper ( double novo )
   {
     valor = novo;
   } // fim do metodo construtor
} // fim da classe e_Wrapper ( )

class Exemplo_10a
{
//
// 2. definir funcao FATORIAL com parametro
   public static int FATORIAL ( int N )
   {
   // 2.1. definir dados
      int X,                        // contador
          PRODUTO = 1;        // produto de naturais
   // 2.2. gerar N numeros naturais
      for ( X = N; X >= 1; X = X-1 )
      {
      // 2.2.1. multiplicar cada numero natural
         PRODUTO = PRODUTO * X;
      } // fim do bloco repetitivo
   // retorno do valor calculado
      return ( PRODUTO );
   } // fim da funcao FATORIAL ( )
```

```
//
// 3. definir procedimento com parametro
   public static void EXP1 ( int N, e_Wrapper E )
   {
   // 3.1. definir dados locais
      int X;                // contador
      double SOMA = 1.0;       // soma de inversos
   // 3.2. gerar (N-1) numeros naturais
      for ( X = 1; X <= (N-1); X = X+1 )
      {
      // 3.2.1. acumular a soma dos inversos
         SOMA = SOMA + 1.0 / FATORIAL ( X );
      } // fim da geracao de (N-1) naturais
   // 3.3 definir valor de retorno na variavel global
      E.valor = SOMA;
   } // fim do procedimento EXP1 ( )
//
// parte principal
//
   public static void main ( String [ ] args )
   {
   // 3. definir dados
      e_Wrapper E = new e_Wrapper ( 0.0 );  // valor de (e) com (x) termos
   // 4. mostrar resultado
      EXP1 ( 10, E );
      System.out.println ( "\ne com 10 termos = " + E.valor );
      EXP1 ( 20, E );
      System.out.println ( "\ne com 20 termos = " + E.valor );
   // pausa para terminar
      System.out.print ( "\nPressionar ENTER para terminar." );
      System.console( ).readLine( );
   } // end main ( )

} // fim Exemplo_10a class
```

Outra versão do programa em Java:

```
/**
  * Exemplo 10b
  * Calcular e mostrar o valor do numero (e) com varias aproximacoes.
  */

// ------------------------------------------------ classes necessarias

// ------------------------------------------------ definicao de classes

// definir um envelope para o objeto
class e_Wrapper
{
// definir o objeto armazenador do dado
   private double valor = 0.0;

// definir como alterar o valor do objeto
   public void putValue ( double novo )
   {
     valor = novo;
   } // fim putValue ( )

// definir como obter o valor do objeto
   public double getValue ( )
   {
     return ( valor );
   } // fim getValue ( )
} // fim da classe e_Wrapper ( )

class Exemplo_10b
{
//
// 2. definir funcao FATORIAL com parametro
   public static int FATORIAL ( int N )
   {
   // 2.1. definir dados
      int X,                          // contador
          PRODUTO = 1;                    // produto de naturais
   // 2.2. gerar N numeros naturais
      for ( X = N; X >= 1; X = X-1 )
      {
      // 2.2.1. multiplicar cada numero natural
         PRODUTO = PRODUTO * X;
      } // fim do bloco repetitivo
   // retorno do valor calculado
      return ( PRODUTO );
   } // fim da funcao FATORIAL ( )
```

```
//
// 3. definir procedimento com parametro
   public static void EXP1 ( int N, e_Wrapper E )
   {
   // 3.1. definir dados locais
      int X;                          // contador
      double SOMA = 1.0;              // soma de inversos
   // 3.2. gerar (N-1) numeros naturais
      for ( X = 1; X <= (N-1); X = X+1 )
      {
      // 3.2.1. acumular a soma dos inversos
         SOMA = SOMA + 1.0 / FATORIAL ( X );
      } // fim da geracao de (N-1) naturais
   // 3.3 definir valor de retorno na variavel global
      E.putValue ( SOMA );
   } // fim do procedimento EXP1 ( )
//
// parte principal
//
   public static void main ( String [ ] args )
   {
   // 3. definir dados
      e_Wrapper E = new e_Wrapper( );  // valor de (e) com (x) termos
   // 4. mostrar resultado
      EXP1 ( 10, E );
      System.out.println ( "\ne com 10 termos = " + E.getValue( ) );
      EXP1 ( 20, E );
      System.out.println ( "\ne com 20 termos = " + E.getValue( ) );
   // pausa para terminar
      System.out.print ( "\nPressionar ENTER para terminar." );
      System.console( ).readLine( );
   } // end main ( )

} // fim Exemplo_10b class
} // fim Exemplo_10b class
```

Programa em Python:

```
# Exemplo 10.
# Calcular e mostrar o valor do numero (e) com varias aproximacoes.
#
# 2. definir a funcao FATORIAL com parametro
def FATORIAL ( N ):
# 2.1. definir dados
  X = 0;            # contador
  PRODUTO = 1;          # produto de naturais
# 2.2. gerar N numeros naturais
  for X in range ( N, 1-1, -1 ):
  # 2.2.1 multiplicar cada numero natural
    PRODUTO = PRODUTO * X;
  # fim do bloco repetitivo
# retorno do valor calculado
  return ( PRODUTO );
# FATORIAL( )
#
# definir procedimento com parametro
def EXP01 ( N, E ):
#
# 2.1 definir dados locais
# int X;                    # contador
  SOMA = 1.0;               # soma de inversos
# 2.2. gerar (N-1) numeros naturais
  for X in range ( 1, (N-1)+1, 1 ):
  # acumular a soma dos inversos
    SOMA = SOMA + 1.0/FATORIAL ( X );
  # fim da geracao de (N-1) naturais
# retornar o valor
  E [0] = SOMA;
# procedimento EXP01( )
#
# parte principal
#
# calcular e mostrar o valor do numero (e) com varias aproximacoes
# definir dado como objeto
E = [0];                    # valor de (e) com (1) termo
# 3. mostrar resultado
EXP01 ( 10, E );
print    ( "\ne com 10 termos = ", E[0] );
EXP01 ( 20, E );
print    ( "\ne com 20 termos = ", E[0] );
# pausa para terminar
print    ( "\nPressionar ENTER para terminar.\n" );
input    ( );
# fim do programa
```

Exercícios

1. Escrever um procedimento com passagem por referência para calcular o fatorial e outro para calcular o somatório abaixo :

$$S(n) = \frac{1}{0!} + \frac{1}{1!} + \frac{1}{2!} + \frac{1}{3!} + \quad \cdots + \frac{1}{n!}$$

recebendo o número de termos (N) como parâmetro.

2. Escrever um procedimento com passagem por referência para calcular o fatorial e uma função para calcular com 20 parcelas :

$$e^x = \frac{x^0}{0!} + \frac{x^1}{1!} + \frac{x^2}{2!} + \frac{x^3}{3!} + \quad \cdots$$

recebendo um valor real (X) como parâmetro.

3. Escrever uma função para calcular o fatorial e um procedimento com passagem por referência para calcular o cosseno de um ângulo (X), em radianos,

$$\cos(x,n) = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \quad \cdots$$

recebendo um valor real (X) como parâmetro, e
para (N) termos, com (N) lido do teclado.

4. Escrever uma função para calcular o fatorial e um procedimento com passagem por referência para gerar o seno de um ângulo (X), em radianos,

$$sen(x,n) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \quad \cdots$$

recebendo um valor real (X) como parâmetro, e
para (N) termos, com (N) lido do teclado.

5. Escrever uma função para a exponenciação, outra para o fatorial e um procedimento com passagem por referência para calcular o somatório :

$$S(x) = \frac{x^{50}}{0!} - \frac{x^{49}}{1!} + \frac{x^{48}}{2!} - \quad \cdots \quad - \frac{x^1}{49!} + \frac{x^0}{50!}$$

recebendo um valor real (X) como parâmetro.

Sugestão: Para fazer a exponenciação considerar as operações abaixo.

$x^y = z = \exp(y * \ln (x))$

Escopo de identificadores

Uma vez que é permitido a definição de objetos locais, tanto no procedimento, quanto na função, pode haver repetição do uso do mesmo nome (identificador) de um objeto externo, ou mesmo de um objeto definido em um outro procedimento, ou função.

Além disso, em outras linguagens, diferente de C++, é possível definir procedimentos, e funções, confinados em outros procedimentos, ou funções, estabelecendo diferentes níveis de definição de objetos. Para evitar ambigüidades, diz-se que a utilização de determinado identificador faz referência ao objeto declarado no mesmo nível, ou no nível mais próximo, partindo-se do atual para o mais externo.

Exemplo 19:

Fazer um procedimento para trocar os valores de duas variáveis, se o segundo for menor que o primeiro.

Análise de dados:

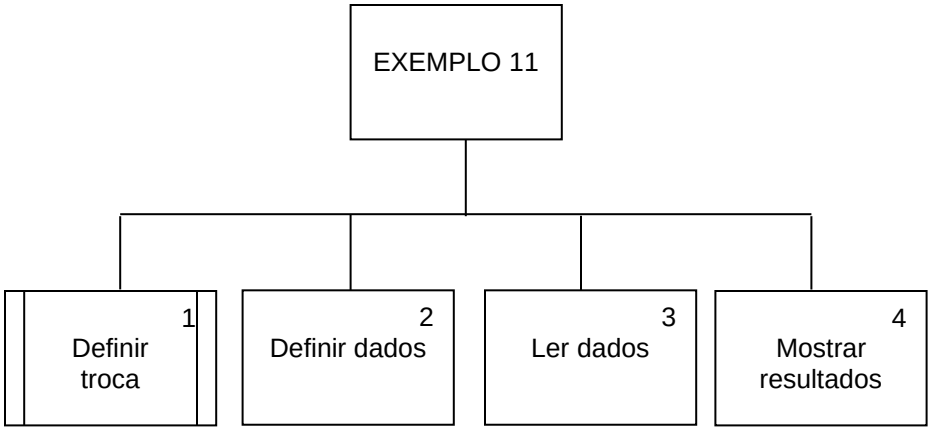
- Dados do procedimento:

Dado	Tipo	Valor Inicial	Obs.
X	inteiro		primeiro valor
Y	inteiro		segundo valor

- Dados da parte principal:

Dado	Tipo	Valor Inicial	Obs.
X	inteiro		primeiro valor
Y	inteiro		segundo valor

Diagrama funcional:



Algoritmo, com procedimento:

Esboço:

Primeira versão, só comentários.

Exemplo 11	v.1
Ação	Bloco
! definir procedimento TROCAR (inteiro &X, inteiro &Y)	1
! definir dados da parte principal	2
! ler dados	3
! mostrar resultado	4

Segunda versão, refinar o segundo e terceiro blocos.

Exemplo 11	v.2
Ação	Bloco
! definir procedimento TROCAR (inteiro &X, inteiro &Y)	1
! definir dados da parte principal inteiro X, ! primeiro valor Y; ! segundo valor	2
! ler dados tela ← ("\nQual o valor de X ? "); X ← teclado; tela ← ("\nQual o valor de Y ? "); Y ← teclado;	3
! mostrar resultado	4
! se o segundo for maior que o primeiro, trocar	4.1
! mostrar X seguido de Y	4.2

Terceira versão, refinar o quarto bloco.

Exemplo 11	v.3
Ação	Bloco
! definir procedimento TROCAR (inteiro &X, inteiro &Y)	1
! definir dados da parte principal inteiro X, ! primeiro valor Y; ! segundo valor	2
! ler dados tela ← ("\nQual o valor de X ? "); X ← teclado; tela ← ("\nQual o valor de Y ? "); Y ← teclado;	3
! mostrar resultado	4
! testar a necessidade de troca	4.1
X > Y ? | V | TROCAR (X, Y);	
! mostrar X seguido de Y	4.2
tela ← ("\n", X, " ", Y);	

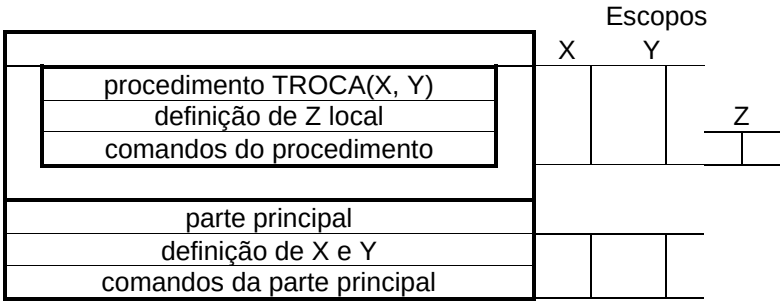
Quarta versão, refinar novamente o quarto bloco.

Exemplo 11	v.3
Ação	Bloco
! definir procedimento TROCAR (inteiro &X, inteiro &Y)	1
! definir dados da parte principal inteiro X, ! primeiro valor Y; ! segundo valor	2
! ler dados tela ← ("\nQual o valor de X ? "); X ← teclado; tela ← ("\nQual o valor de Y ? "); Y ← teclado;	3
! mostrar resultado	4
! testar a necessidade de troca	4.1
se (X > Y) TROCAR (X, Y); fim se	
! mostrar X seguido de Y	4.2
tela ← ("\n", X, " ", Y);	

Quinta versão, refinar o primeiro bloco.

Exemplo 11	v.3
Ação	Bloco
! definir procedimento TROCAR (inteiro &X, inteiro &Y)	1
procedimento TROCAR (inteiro & X, inteiro & Y)	
! definir dado local	1.1
inteiro Z; ! valor auxiliar para troca	
! trocar X com Y usando Z	1.2
Z = X; ! guardar o valor do primeiro X = Y; ! atualizar com o valor do segundo Y = Z; ! recuperar o valor guardado	
! definir dados da parte principal inteiro X, ! primeiro valor Y; ! segundo valor	2
! ler dados tela ← ("\nQual o valor de X ? "); X ← teclado; tela ← ("\nQual o valor de Y ? "); Y ← teclado;	3
! mostrar resultado	4
! verificar a necessidade de troca	4.1
se (X > Y) TROCAR (X, Y); fim se	
! mostrar X seguido de Y	4.2
tela ← ("\n", X, " ", Y);	

Em um diagrama de escopos:



No exemplo acima, os nomes X e Y foram utilizados tanto no procedimento, como na parte principal do algoritmo. Entretanto, os identificadores definidos no procedimento não são visíveis na parte principal, e vice-versa. Apenas o mecanismo de passagem de parâmetros por referência é responsável por fazer corresponder os primeiros aos segundos.

Programa em SCILAB:

```
// Exemplo 11.
// Ler dois valores do teclado e mostrar o menor antes do maior.
//
// definir dados globais (X,Y)
  global X;
  global Y;            // valores a serem comparados
//
// 1. definir procedimento para trocar valores
function TROCAR ( X, Y )
//
  global X;
  global Y;
// 1.1. definir dado local
  Z = 0;               // valor auxiliar para troca
// 1.2. trocar X com Y usando Z
  Z = X;               // guardar o valor do primeiro
  X = Y;               // atualizar com o valor do segundo
  Y = Z;               // recuperar o valor guardado
endfunction // TROCAR( )
//
// parte principal
//
// ler dois valores inteiros e mostrar o menor antes do maior;
// se necessario, trocar valores
// 2. definir dados da parte principal
  global X;
  global Y;
  X = 0;     // primeiro valor
  Y = 0;     // segundo valor
// 3. ler dados
  clc;   // limpar a area de trabalho
  X = input ( "\nQual o valor de X ? " );
  Y = input ( "\nQual o valor de Y ? " );
// 4. mostrar resultado
// 4.1 verificar a necessidade de troca
  if ( X > Y )
    TROCAR ( X, Y );
  end // if ( X > Y )
// 4.2 mostrar X seguido de Y
  printf ( "\n %d %d", X, Y );
// pausa para terminar
  printf ( "\nPressionar ENTER para terminar.\n" );
  halt;
// fim do programa
```

Programa em C:

```
// Exemplo 11.
// Ler dois valores do teclado e mostrar o menor antes do maior.
//
// bibliotecas necessarias
#include <iostream>
using namespace std;
//
// 1. definir procedimento para trocar valores
void TROCAR ( int & X, int & Y )
{
 // 1.1. definir dado local
   int Z,                  // valor auxiliar para troca
 // 1.2. trocar X com Y usando Z
   Z = X;                  // guardar o valor do primeiro
   X = Y;                  // atualizar com o valor do segundo
   Y = Z;                  // recuperar o valor guardado
} // fim TROCAR( )
//
// parte principal
//
int main (void)
{
// 2. definir dados da parte principal
   int X,                  // primeiro valor
       Y;                 // segundo valor
// 3. ler dados
   printf ( "\nQual o valor de X ? " );   scanf ( "%d", &X );
   printf ( "\nQual o valor de Y ? " );   scanf ( "%d", &Y );
// 4. mostrar resultado
// 4.1 verificar a necessidade de troca
   if ( X > Y )
   {
     TROCAR ( X, Y );
   } // fim se ( X > Y )
// 4.2 mostrar X seguido de Y
   printf ( "\n%d%s%d", X, " ", Y );
// pausa para terminar
   printf ( "Pressionar ENTER para terminar." );
   getchar ( );
   return ( 0 );
} // fim do programa
```

Programa em C++:

```
// Exemplo 11.
// Ler dois valores do teclado e mostrar o menor antes do maior.
//
// bibliotecas necessarias
#include <iostream>
using namespace std;
//
// 1. definir procedimento para trocar valores
void TROCAR ( int & X, int & Y )
{
 // 1.1. definir dado local
   int Z,                  // valor auxiliar para troca
 // 1.2. trocar X com Y usando Z
   Z = X;                  // guardar o valor do primeiro
   X = Y;                  // atualizar com o valor do segundo
   Y = Z;                  // recuperar o valor guardado
} // fim TROCAR( )
//
// parte principal
//
int main (void)
{
// 2. definir dados da parte principal
   int X,                  // primeiro valor
       Y;                 // segundo valor
// 3. ler dados
   cout << "\nQual o valor de X ? ";   cin >> X;
   cout << "\nQual o valor de Y ? ";   cin >> Y;
// 4. mostrar resultado
// 4.1 verificar a necessidade de troca
   if ( X > Y )
   {
      TROCAR ( X, Y );
   } // fim se ( X > Y )
// 4.2 mostrar X seguido de Y
   cout << "\n" << X << " " <<  Y;
// pausa para terminar
   cout << "Pressionar ENTER para terminar.";
   cin.get ( );
   return EXIT_SUCCESS;
} // fim do programa
```

Programa em C#:

```
/*
 * Exemplo 11
 * Ler dois valores do teclado e mostrar o menor antes do maior.
 */
using System;

class Exemplo_11
{
// 1. definir procedimento para trocar valores
   public static void TROCAR ( ref int X, ref int Y )
   {
   // 1.1. definir dado local
      int Z;                    // valor auxiliar para troca
   // 1.2. trocar X com Y usando Z
      Z = X;                    // guardar o valor do primeiro
      X = Y;                    // atualizar com o valor do segundo
      Y = Z;                    // recuperar o valor guardado
//    Console.WriteLine ( "\nem TROCAR: "+ X +" "+ Y );
   } // fim TROCAR ( )
//
// parte principal
//
   public static void Main ( )
   {
   // 2. definir dados
      int X = 0,  // primeiro  valor
          Y = 0;  // segundo valor
   // 3. ler dados
      Console.Write ("\nQual o valor de X ? " );
      X = int.Parse ( Console.ReadLine ( ) ); // ler primeiro valor
      Console.Write ("\nQual o valor de Y ? " );
      Y = int.Parse ( Console.ReadLine ( ) ); // ler segundo valor
   // 4. mostrar resultado
   // 4.1. verificar a necessidade de troca
      if ( X > Y )
      {
        TROCAR ( ref X, ref Y );
      }
   // 4.2 mostrar X seguido de Y
      Console.WriteLine ( "\n" + X + " " + Y );
   // pausa para terminar
      Console.Write ( "\nPressionar ENTER para terminar." );
      Console.ReadLine ( );
   } // end Main ( )

} // fim Exemplo_11 class
```

Programa em Java:

```
/**
  * Exemplo 11
  * Ler dois valores do teclado e mostrar o menor antes do maior.
  */

// ------------------------------------------------ classes necessarias
import IO.*;                                  // IO.jar   deve ser acessivel
// ------------------------------------------------ definicao de classes

// definir um envelope para o objeto
class int_Wrapper
{
// definir conversao para caracteres
   public String toString ( )
   {
     return ( "" + valor );
   } // fim toString ( )

// definir o objeto armazenador do dado
   private int valor;

// definir um construtor padrao para o objeto
   public int_Wrapper ( int inicial )
   {
     valor = inicial;
   } // fim construtor padrao

// definir um construtor alternativo para o objeto
   public int_Wrapper ( )
   {
     valor = 0;
   } // fim construtor alternativo

// definir como alterar o valor do objeto
   public void putValue ( int novo )
   {
     valor = novo;
   } // fim putValue ( )

// definir como obter o valor do objeto
   public int getValue ( )
   {
     return ( valor );
   } // fim getValue ( )

// definir como comparar o valor do objeto
   public int compareTo ( int_Wrapper outro )
   {
     return ( valor - outro.getValue ( ) );
   } // fim compareTo ( )

} // fim da classe int_Wrapper ( )
```

```
class Exemplo_11
{
// 1. definir procedimento para trocar valores
   public static void TROCAR ( int_Wrapper X, int_Wrapper Y )
   {
   // 1.1. definir dado local
      int Z;                // valor auxiliar para troca
   // 1.2. trocar X com Y usando Z
      Z = X.getValue ( );         // guardar o valor do primeiro
                                  // atualizar com o valor do segundo
      X.putValue ( Y.getValue( ) );
                                  // recuperar o valor guardado
      Y.putValue ( Z );
//      IO.println ( "\nem TROCAR: "+X.intValue( )+" "+Y.intValue( ) );
   } // fim TROCAR ( )
//
// parte principal
//
   public static void main ( String [ ] args )
   {
   // 2. definir dados
      int_Wrapper X = null,  // primeiro valor
                  Y = null;  // segundo  valor
   // 3. ler dados
      X = new int_Wrapper ( IO.readint ( "\nQual o valor de X ? " ) );
      Y = new int_Wrapper ( IO.readint ( "\nQual o valor de Y ? " ) );
   // 4. mostrar resultado
   // 4.1. verificar a necessidade de troca
      if ( X.compareTo ( Y ) > 0 )
      {
        TROCAR ( X, Y );
      }
   // 4.2 mostrar X seguido de Y
      IO.println ( "\n" + X + " " + Y );
   // pausa para terminar
      IO.pause ( "\nPressionar ENTER para terminar." );
   } // end main ( )

} // fim Exemplo_11 class
```

Programa em Python:

```
# Exemplo 11.
# Ler dois valores do teclado e mostrar o menor antes do maior.
#
# definir dados globais (X,Y)
X = 0;
Y = 0;              # valores a serem comparados
#
# 1. definir procedimento para trocar valores
def TROCAR ( ):
#
  global X;
  global Y;
# 1.1. definir dado local
  Z = 0;            # valor auxiliar para troca
# 1.2. trocar X com Y usando Z
  Z = X;            # guardar o valor do primeiro
  X = Y;            # atualizar com o valor do segundo
  Y = Z;            # recuperar o valor guardado
# TROCAR( )
#
# parte principal
#
# ler dois valores inteiros e mostrar o menor antes do maior;
# se necessario, trocar valores
# 2. definir dados da parte principal
X = 0;   # primeiro valor
Y = 0;   # segundo  valor
# 3. ler dados
X = int ( input ( "\nQual o valor de X ? " ) );
Y = int ( input ( "\nQual o valor de Y ? " ) );
# 4. mostrar resultado
# 4.1 verificar a necessidade de troca
if ( X > Y ):
  TROCAR ( );
# if ( X > Y )
# 4.2 mostrar X seguido de Y
print  ( "\n X=%d Y=%d"%( X, Y ) );
# pausa para terminar
print  ( "\nPressionar ENTER para terminar.\n" );
input ( );
# fim do programa
```

Outra versão do programa em Python:

```
# Exemplo 11b.
# Ler dois valores do teclado e mostrar o menor antes do maior.
#
from copy import copy
#
# 1. definir procedimento para trocar valores
def TROCAR ( X, Y ):
#
# 1. trocar X com Y usando o metodo copy ( )
   return ( copy(Y), copy(X) );
# TROCAR( )
#
# parte principal
#
# ler dois valores inteiros e mostrar o menor antes do maior;
# se necessario, trocar valores
# 2. definir dados da parte principal
X = 0;    # primeiro valor
Y = 0;    # segundo  valor
# 3. ler dados
X = int ( input ( "\nQual o valor de X ? " ) );
Y = int ( input ( "\nQual o valor de Y ? " ) );
# 4. mostrar resultado
# 4.1 verificar a necessidade de troca
if ( X > Y ):
   X, Y = TROCAR ( X, Y );
# if ( X > Y )
# 4.2 mostrar X seguido de Y
print  ( "\n X=%d Y=%d"%( X, Y ) );
# pausa para terminar
print  ( "\nPressionar ENTER para terminar.\n" );
input ( );
# fim do programa
```

Exercícios

1. Fazer um algoritmo para:
   - ler dois números inteiros através de um procedimento com passagens por referências, que garanta que ambos serão positivos;
   - usando o procedimento de troca, mostrar qual o maior deles.

2. Fazer um algoritmo para:
   - ler dois números reais através de um procedimento com passagens por referências, que garanta que ambos serão positivos;
   - usando o procedimento de troca, mostrar qual o menor deles.

3. Fazer um algoritmo para:
   - ler dois números inteiros através de um procedimento com passagens por referências, que garanta que ambos serão positivos, e que o primeiro será menor que o segundo;
   - usando os valores fornecidos pelo procedimento acima, definir um outro procedimento para ler um valor inteiro pertencente ao intervalo entre os dois primeiros, que também deverão ser passados como parâmetros, mas apenas por valor.

4. Fazer um algoritmo para:
   - ler dois números inteiros através de um procedimento com passagens por referências, que garanta que ambos serão positivos, e que o primeiro será menor que o segundo;
   - usando os valores fornecidos pelo procedimento acima, definir um outro procedimento para ler um valor inteiro que não pertença ao intervalo entre os dois primeiros, que também deverão ser passados como parâmetros, mas apenas por valor.

5. Fazer um algoritmo para:
   - definir um procedimento para ler um caractere no intervalo fechado definido por outras duas letras que também deverão ser passadas como parâmetros, mas apenas por valor.

Exemplo 12:

Fazer um procedimento para colocar três valores inteiros em ordem, usando o mecanismo de trocas.

Análise de dados:

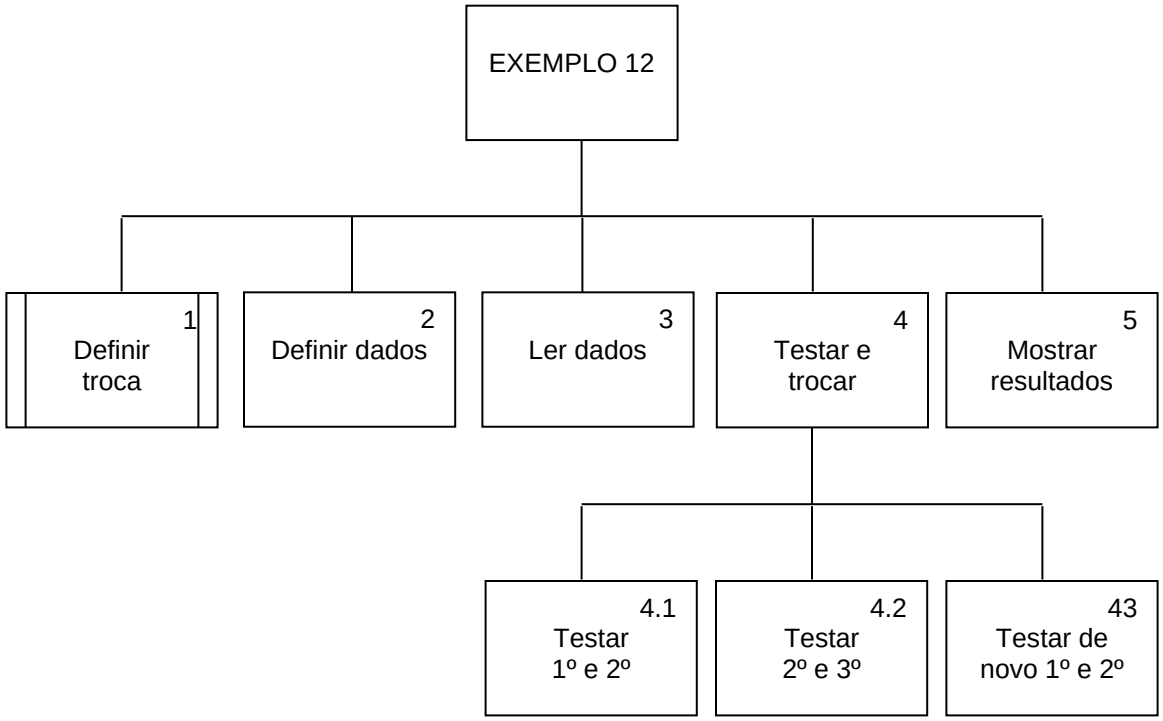
- Dados do procedimento:

Dado	Tipo	Valor Inicial	Obs.
X	inteiro		primeiro valor
Y	inteiro		segundo valor

- Dados da parte principal:

Dado	Tipo	Valor Inicial	Obs.
X	inteiro		primeiro valor
Y	inteiro		segundo valor
Z	inteiro		terceiro valor

Diagrama funcional:



Algoritmo, com procedimento:

Esboço:

Primeira versão, só comentários.

Exemplo 12	v.1
Ação	Bloco
! definir procedimento TROCAR (inteiro &X, inteiro &Y)	1
! definir dados da parte principal	2
! ler dados	3
! testar e trocar	4
! testar o 1º e o 2º	4.1
! testar o 2º e o 3º	4.2
! testar o 1º e o 2º, de novo	4.3
! mostrar resultado	5

Segunda versão, refinar o segundo e terceiro blocos.

Exemplo 12	v.2
Ação	Bloco
! definir procedimento TROCAR (inteiro &X, inteiro &Y)	1

! definir dados da parte principal
  inteiro X, ! primeiro valor
  Y, ! segundo valor
  Z; ! terceiro valor
	2

! ler dados
  tela ← ("\nQual o valor de X ? "); X ← teclado;
  tela ← ("\nQual o valor de Y ? "); Y ← teclado;
  tela ← ("\nQual o valor de Z ? "); Z ← teclado;
	3

! testar e trocar	4
! testar o 1º e o 2º	4.1
! testar o 2º e o 3º	4.2
! testar o 1º e o 2º, de novo	4.3
! mostrar resultado	5

Terceira versão, refinar o quinto bloco.

Exemplo 12	v.3
Ação	Bloco
! definir procedimento TROCAR (inteiro &X, inteiro &Y)	1

! definir dados da parte principal
  inteiro X, ! primeiro valor
  Y, ! segundo valor
  Z; ! terceiro valor
	2

! ler dados
  tela ← ("\nQual o valor de X ? "); X ← teclado;
  tela ← ("\nQual o valor de Y ? "); Y ← teclado;
  tela ← ("\nQual o valor de Z ? "); Z ← teclado;
	3

! testar e trocar	4
! testar o 1º e o 2º	4.1
! testar o 2º e o 3º	4.2
! testar o 1º e o 2º, de novo	4.3
! mostrar resultado tela ← ("\n", X, " ", Y, " ", Z);	5

Quarta versão, refinar o quarto bloco.

Exemplo 12			v.4
Ação			Bloco
! definir procedimento TROCAR (inteiro &X, inteiro &Y)			1
! definir dados da parte principal inteiro X, ! primeiro valor Y, ! segundo valor Z; ! terceiro valor			2
! ler dados tela ← ("\nQual o valor de X ? "); X ← teclado; tela ← ("\nQual o valor de Y ? "); Y ← teclado; tela ← ("\nQual o valor de Z ? "); Z ← teclado;			3
! testar e trocar			4
! testar o 1º e o 2º			4.1
X > Y ?	V	TROCAR (X, Y);	
! testar o 2º e o 3º			4.2
Y > Z ?	V	TROCAR (Y, Z);	
! testar o 1º e o 2º, de novo			4.3
X > Y ?	V	TROCAR (X, Y);	
! mostrar resultado tela ← ("\n", X, " ", Y, " ", Z);			5

Quinta versão, refinar novamente o quarto bloco.

Exemplo 12	v.5
Ação	Bloco
! definir procedimento TROCAR (inteiro &X, inteiro &Y)	1
! definir dados da parte principal inteiro X, ! primeiro valor Y, ! segundo valor Z; ! terceiro valor	2
! ler dados tela ← ("\nQual o valor de X ? "); X ← teclado; tela ← ("\nQual o valor de Y ? "); Y ← teclado; tela ← ("\nQual o valor de Z ? "); Z ← teclado;	3
! testar e trocar	4
! testar o 1º e o 2º	4.1
se (X > Y) TROCAR (X, Y); fim se	
! testar o 2º e o 3º	4.2
se (Y > Z) TROCAR (Y, Z); fim se	
! testar o 1º e o 2º, de novo	4.3
se (X > Y) TROCAR (X, Y); fim se	
! mostrar resultado tela ← ("\n", X, " ", Y, " ", Z);	5

Sexta versão, refinar o primeiro bloco.

Exemplo 12	v.6
Ação	Bloco
! definir procedimento TROCAR (inteiro &X, inteiro &Y)	1
procedimento TROCAR (inteiro & X, inteiro & Y)	
! definir dado local	1.1
inteiro Z; ! valor auxiliar para troca	
! trocar X com Y usando Z	1.2
Z = X; ! guardar o valor do primeiro X = Y; ! atualizar com o valor do segundo Y = Z; ! recuperar o valor guardado	
! parte principal	
! definir dados da parte principal inteiro X, ! primeiro valor Y, ! segundo valor Z; ! terceiro valor	2
! ler dados tela ← ("\nQual o valor de X ? "); X ← teclado; tela ← ("\nQual o valor de Y ? "); Y ← teclado; tela ← ("\nQual o valor de Z ? "); Z ← teclado;	3
! testar e trocar	4
! testar o 1º e o 2º	4.1
se (X > Y) TROCAR (X, Y); fim se	
! testar o 2º e o 3º	4.2
se (Y > Z) TROCAR (Y, Z); fim se	
! testar o 1º e o 2º, de novo	4.3
se (X > Y) TROCAR (X, Y); fim se	
! mostrar resultado tela ← ("\n", X, " ", Y, " ", Z);	5

Programa em SCILAB:

```
// Exemplo 12.
// Ler tres valores inteiros e mostra-los em ordem crescente.
//
// 1. definir procedimento para trocar valores
function [ A, B ] = TROCAR ( X, Y )
//
// 1.1. definir dado local
  Z = 0;                // valor auxiliar para troca
// 1.2. trocar X com Y usando Z
  Z = X;                // guardar o valor do primeiro
  X = Y;                // atualizar com o valor do segundo
  Y = Z;                // recuperar o valor guardado
// retornos
  A = X;
  B = Y;
endfunction // TROCAR( )
//
// parte principal
//
// Ler tres valores inteiros e mostra-los em ordem crescente.
// 2. definir dados da parte principal
  X = 0;     // primeiro  valor
  Y = 0;     // segundo valor
  Z = 0;     // terceiro   valor
// 3. ler dados
  clc;   // limpar a area de trabalho
  X = input ( "\nQual o valor de X ? " );
  Y = input ( "\nQual o valor de Y ? " );
  Z = input ( "\nQual o valor de Z ? " );
// 4. testar e trocar
// 4.1 testar o 1º e o 2º valores
  if ( X > Y )
    [ X, Y ] = TROCAR ( X, Y );
  end
// 4.2 testar o 2º e o 3º valores
  if ( Y > Z )
    [ Y, Z ] = TROCAR ( Y, Z );
  end
// 4.3 testar o 1º e o 2º valores, de novo
  if ( X > Y )
    [ X, Y ] = TROCAR ( X, Y );
  end
// 5. mostrar resultado
  printf ( "\n %d %d %d", X, Y, Z );
// pausa para terminar
  printf ( "\nPressionar ENTER para terminar.\n" );
  halt;
// fim do programa
```

Programa em C:

```
// Exemplo 12.
// Ler tres valores inteiros e mostra-los em ordem crescente.
//
// bibliotecas necessarias
#include <iostream>
using namespace std;
//
// 1. definir procedimento para trocar valores
void TROCAR ( int *X, int *Y )
{
 // 1.1. definir dado local
   int Z,                  // valor auxiliar para troca
 // 1.2. trocar X com Y usando Z
    Z = *X;                // guardar o valor do primeiro
   *X = *Y;                // atualizar com o valor do segundo
   *Y = Z;                 // recuperar o valor guardado
} // fim do procedimento EXP1
//
// parte principal
//
int main (void)
{
// 2. definir dados da parte principal
   int X,                  // primeiro valor
       Y,                  // segundo valor
       Z;                  //  terceiro  valor
// 3. ler dados
   printf ( "\nQual o valor de X ? " );   scanf ( "%d", &X );
   printf ( "\nQual o valor de Y ? " );   scanf ( "%d", &Y );
   printf ( "\nQual o valor de Z ? " );   scanf ( "%d", &Z );
// 4. testar e trocar
// 4.1 testar o 1º e o 2º valores
   if ( X > Y )
   { TROCAR ( X, Y ); }
// 4.2 testar o 2º e o 3º valores
   if ( Y > Z )
   { TROCAR ( Y, Z ); }
// 4.3 testar o 1º e o 2º valores, de novo
   if ( X > Y )
   { TROCAR ( X, Y ); }
// 5. mostrar resultado
   printf ( "\n%d%s%d%s%d", X, " ", Y, " ", Z );
// pausa para terminar
   printf ( "Pressionar ENTER para terminar." );
   getchar ( );
   return ( 0 );
} // fim do programa
```

Programa em C++:

```
// Exemplo 12.
// Ler tres valores inteiros e mostra-los em ordem crescente.
//
// bibliotecas necessarias
#include <iostream>
using namespace std;
//
// 1. definir procedimento para trocar valores
void TROCAR ( int & X, int & Y )
{
 // 1.1. definir dado local
   int Z,                  // valor auxiliar para troca
 // 1.2. trocar X com Y usando Z
   Z = X;                  // guardar o valor do primeiro
   X = Y;                  // atualizar com o valor do segundo
   Y = Z;                  // recuperar o valor guardado
} // fim do procedimento EXP1
//
// parte principal
//
int main (void)
{
// 2. definir dados da parte principal
   int X,                  // primeiro valor
       Y,                 // segundo valor
       Z;                 //  terceiro  valor
// 3. ler dados
   cout << "\nQual o valor de X ? ";   cin >> X;
   cout << "\nQual o valor de Y ? ";   cin >> Y;
   cout << "\nQual o valor de Z ? ";   cin >> Z;
// 4. testar e trocar
// 4.1 testar o 1º e o 2º valores
   if ( X > Y )
   { TROCAR ( X, Y ); }
// 4.2 testar o 2º e o 3º valores
   if ( Y > Z )
   { TROCAR ( Y, Z ); }
// 4.3 testar o 1º e o 2º valores, de novo
   if ( X > Y )
   { TROCAR ( X, Y ); }
// 5. mostrar resultado
   cout << "\n" << X << " " <<  Y << " " << Z;
// pausa para terminar
   cout << "Pressionar ENTER para terminar.";
   cin.get ( );
   return EXIT_SUCCESS;
} // fim do programa
```

Programa em C#:

```
/*
 * Exemplo 12
 * Ler tres valores do teclado e mostra-los em ordem crescente.
 */
using System;

class Exemplo_12
{
// 1. definir procedimento para trocar valores
   public static void TROCAR ( ref int X, ref int Y )
   {
   // 1.1. definir dado local
      int Z;                    // valor auxiliar para troca
   // 1.2. trocar X com Y usando Z
      Z = X;                    // guardar o valor do primeiro
      X = Y;                    // atualizar com o valor do segundo
      Y = Z;                    // recuperar o valor guardado

//    Console.WriteLine ( "\nem TROCAR: "+ X +" "+ Y );
   } // fim TROCAR ( )
//
// parte principal
//
   public static void Main ( )
   {
   // 2. definir dados
      int X = 0,  // primeiro  valor
          Y = 0,  // segundo valor
          Z = 0;  // terceiro   valor
   // 3. ler dados
      Console.Write ("\nQual o valor de X ? " );
      X = int.Parse ( Console.ReadLine ( ) ); // ler primeiro valor
      Console.Write ("\nQual o valor de Y ? " );
      Y = int.Parse ( Console.ReadLine ( ) ); // ler segundo valor
      Console.Write ("\nQual o valor de Z ? " );
      Z = int.Parse ( Console.ReadLine ( ) ); // ler terceiro   valor
   // 4. testar e trocar
   // 4.1 testar o 1º e o 2º valores
      if ( X > Y )   { TROCAR ( ref X, ref Y ); }
   // 4.2 testar o 2º e o 3º valores
      if ( Y > Z )   { TROCAR ( ref Y, ref Z ); }
   // 4.3 testar o 1º e o 2º valores, de novo
      if ( X > Y )   { TROCAR ( ref X, ref Y ); }
// 5. mostrar resultado
      Console.WriteLine ( "\n" + X + " " + Y +  " " + Z );
   // pausa para terminar
      Console.Write ( "\nPressionar ENTER para terminar." );
      Console.ReadLine ( );
   } // end Main ( )

} // fim Exemplo_12 class
```

Programa em Java:

```
/**
  * Exemplo 12
  * Ler tres valores do teclado e mostra-los em ordem crescente.
  */

// ---------------------------------------------- classes necessarias
import IO.*;                                  // IO.jar   deve ser acessivel
// ---------------------------------------------- definicao de classes

// definir um envelope para o objeto
class int_Wrapper
{
// definir conversao para caracteres
   public String toString ( )
   {
     return ( "" + valor );
   } // fim toString ( )

// definir o objeto armazenador do dado
   private int valor;

// definir um construtor padrao para o objeto
   public int_Wrapper ( int inicial )
   {
     valor = inicial;
   } // fim construtor padrao

// definir um construtor alternativo para o objeto
   public int_Wrapper ( )
   {
     valor = 0;       // valor arbitrario
   } // fim construtor alternativo

// definir como alterar o valor do objeto
   public void putValue ( int novo )
   {
     valor = novo;
   } // fim putValue ( )

// definir como obter o valor do objeto
   public int getValue ( )
   {
     return ( valor );
   } // fim getValue ( )

// definir como comparar o valor do objeto
   public int compareTo ( int_Wrapper outro )
   {
     return ( valor - outro.getValue ( ) );
   } // fim compareTo ( )

} // fim da classe int_Wrapper ( )
```

```
// ---------------------------------------------

      class Exemplo_12
      {
      // 1. definir procedimento para trocar valores
        static void TROCAR ( int_Wrapper X, int_Wrapper Y )
        {
        // 1.1. definir dado local
          int Z;                    // valor auxiliar para troca
        // 1.2. trocar X com Y usando Z
          Z = X.getValue ( );       // guardar o valor do primeiro
                                    // atualizar com o valor do segundo
          X.putValue ( Y.getValue( ) );
                                    // recuperar o valor guardado
          Y.putValue ( Z );
      //    IO.println ( "\nem TROCAR: "+X.intValue()+" "+Y.intValue() );
        } // fim TROCAR ( )
      //
      // parte principal
      //
        public static void main ( String [ ] args )
        {
        // 2. definir dados
          int_Wrapper X = null,  // primeiro valor
                      Y = null,  // segundo  valor
                      Z = null;  // terceiro valor
        // 3. ler dados
          X = new int_Wrapper ( IO.readint ( "\nQual o valor de X ? " ) );
          Y = new int_Wrapper ( IO.readint ( "\nQual o valor de Y ? " ) );
          Z = new int_Wrapper ( IO.readint ( "\nQual o valor de Z ? " ) );
        // 4. testar e trocar
        // 4.1. testar o 1o. e o 2o. valores
          if ( X.compareTo ( Y ) > 0 )
          { TROCAR ( X, Y ); }
        // 4.2. testar o 2o. e o 3o. valores
          if ( Y.compareTo ( Z ) > 0 )
          { TROCAR ( Y, Z ); }
        // 4.3. testar o 1o. e o 2o. valores, de novo
          if ( X.compareTo ( Y ) > 0 )
          { TROCAR ( X, Y ); }
        // 4.2 mostrar X seguido de Y
          IO.println ( "\n" + X + " " + Y + " " + Z );
        // pausa para terminar
          IO.pause ( "\nPressionar ENTER para terminar." );
        } // end main ( )

      } // fim Exemplo_12 class
```

Programa em Python:

```
# Exemplo 12.
# Ler tres valores inteiros e mostra-los em ordem crescente.
#
# 1. definir procedimento para trocar valores
def TROCAR ( X, Y ):
#
# 1.1. definir dado local
   Z = 0;            # valor auxiliar para troca
# 1.2. trocar X com Y usando Z
   Z = X;            # guardar o valor do primeiro
   X = Y;            # atualizar com o valor do segundo
   Y = Z;            # recuperar o valor guardado
# retornos
   A = X;
   B = Y;
   return ( A, B );
# TROCAR( )
#
# parte principal
#
# Ler tres valores inteiros e mostra-los em ordem crescente.
# 2. definir dados da parte principal
X = 0;   # primeiro  valor
Y = 0;   # segundo valor
Z = 0;   # terceirovalor
# 3. ler dados
X = int ( input ( "\nQual o valor de X ? " ) );
Y = int ( input ( "\nQual o valor de Y ? " ) );
Z = int ( input ( "\nQual o valor de Z ? " ) );
# 4. testar e trocar
# 4.1 testar o primeiro e o segundo valor
if ( X > Y ):
   X, Y = TROCAR ( X, Y );
# fim se
# 4.2 testar o segundo  e o terceiro valor
if ( Y > Z ):
   Y, Z = TROCAR ( Y, Z );
# fim se
# 4.3 testar o primeiro e o segundo  valor, de novo
if ( X > Y ):
   X, Y = TROCAR ( X, Y );
# fim se
# 5. mostrar resultado
print ( "\n ", X, " ", Y, " ", Z );
# pausa para terminar
print ( "\nPressionar ENTER para terminar.\n" );
input ( );
# fim do programa
```

Exercícios

1. Fazer um algoritmo para:
   - ler três caracteres;
   - usando o procedimento de troca, colocá-los em ordem crescente.

2. Fazer um algoritmo para:
   - ler quatro valores inteiros;
   - usando o procedimento de troca, colocá-los em ordem crescente.

3. Fazer um algoritmo para:
   - ler quatro valores reais;
   - usando o procedimento de troca, colocá-los em ordem decrescente.

4. Fazer um algoritmo para:
   - definir um procedimento para ler três valores inteiros;
   - usando um procedimento de troca, garantir que o primeiro valor esteja no intervalo fechado definido pelos outros dois.

5. Fazer um algoritmo para:
   - definir um procedimento para ler caracteres;
   - usando um procedimento de troca, garantir que o último valor esteja no intervalo fechado definido pelos outros dois.

Exercícios propostos:

1. Escrever uma função para calcular o valor absoluto de um número inteiro.

   Exemplo para aplicação:

   cout << ABS (X);

   usar a função para calcular a soma de N valores inteiros lidos do teclado.

2. Escrever uma função para calcular :

$$\pi = 4 - \frac{4}{3} + \frac{4}{5} - \frac{4}{7} + \cdots$$

   recebendo o número de parcelas (N) como parâmetro.
   Mostrar o resultado para vários valores de (N) lidos do teclado,
   o último valor terá N = 1.

3. Escrever uma função para verificar se um caractere é uma letra maiúscula.
   Usar a função para testar (N) símbolos lidos do teclado, um por vez.

4. Escrever uma função para verificar se um caractere é uma letra minúscula.
   Usar a função para testar (N) símbolos lidos do teclado, um por vez.

5. Escrever uma função para verificar se um caractere é uma letra (maiúscula ou minúscula).
   Usar a função para testar (N) símbolos lidos do teclado, um por vez.

6. Escrever uma função para verificar se um caractere é um algarismo entre 0 e 9.
   Usar a função para testar (N) símbolos lidos do teclado, um por vez.

7. Escrever uma função para verificar se um caractere é um espaço em branco.
   Usar a função para testar (N) símbolos lidos do teclado, um por vez.

8. Escrever uma função para verificar se um caractere é um sinal de pontuação { . , ; : }
   Usar a função para testar (N) símbolos lidos do teclado, um por vez.

9. Escrever uma função para verificar se um caractere é um operador aritmético { - + * / // }
   Usar a função para testar (N) símbolos lidos do teclado, um por vez.

10. Escrever uma função para verificar se um caractere é um operador lógico { & | ! ^ }
    Usar a função para testar (N) símbolos lidos do teclado, um por vez.

11. Considerando a função abaixo:

```
real função MAIOR (real X,Y)
|   real VALOR
|   se X > Y então VALOR ← X fim se ! X > Y !
|   se X = Y então VALOR ← X fim se ! X = Y !
|   se X < Y então VALOR ← Y fim se ! X < Y !
|   retornar (VALOR)
fim função ! MAIOR !
```

Calcular o resultado para as seguintes chamadas :

a.) MAIOR (4, 5)
b.) MAIOR (6, 4)
c.) MAIOR (5, 5)
d.) MAIOR (12 ***mod*** 3, 15 ***div*** 3)
e.) MAIOR (MAIOR (12 ***mod*** 3,1 5 ***div*** 3),
MAIOR (ABS(12-27+5), 4))

12. Considerando o procedimento abaixo :

```
procedimento S (inteiro & X)
| inteiro T
|
|   T ← 1
|   T ← T + 2 * X
|   X ← T
|
fim procedimento ! S !
```

mostrar o que será calculado pelo trecho :

```
inteiro X1, X2, X3
X1 ← 2
S (X1)
X2 ← 3 + X1
S (X2)
X3 ← X2
S (X3)
tela ← (X1, X2, X3)
```

13. Fazer um procedimento para calcular e mostrar a soma dos 50 primeiros números pares.

14. Fazer um procedimento para calcular e mostrar a soma dos ímpares entre 100 e 500.

15. Fazer um procedimento para calcular e mostrar a soma dos quadrados dos números múltiplos de 3 menores que 100.

16. Fazer um algoritmo para ler um número inteiro (N) do teclado, e calcular e mostrar a soma de N primeiros múltiplos de 2 e 3, por meio de um procedimento.

17. Fazer um algoritmo para ler um número inteiro (N) do teclado, e calcular e mostrar os divisores deste número, por meio de um procedimento.

18. Pode-se calcular a raiz quadrada de um número positivo através do método de aproximação sucessivas de Newton, descrito a seguir:

- seja "a" o número do qual deseja-se obter a raiz quadrada;
- a primeira aproximação para a raiz quadrada será dada por:

$$x_1 = a / 2$$

- a próxima ou sucessiva aproximação é dada por :

$$x_{n+1} = \frac{(x_n^2 + a)}{2x_n}$$

- fazer um algoritmo para :
- ler o valor de "a" do teclado;
- calcular e mostrar a 25ª. aproximação.

19. A conversão de graus Farenheit para Centígrados é obtida por:

$$C = 5 (F - 32) / 9$$

- fazer uma função para calcular e mostrar uma tabela de graus Centígrados em função de graus Farenheit, que variem de 50 a 150 de 1 em 1.

20. Fazer um procedimento para gerar e mostrar a seguinte seqüência:

289 - 256 - 225 - 196 -...- 9 - 4 - 1

21. Fazer um procedimento para calcular e mostrar o valor do somatório abaixo:

$$S = 1 + 3/2 + 5/3 + 7/4 +...+ 99/50$$

22. Fazer um algoritmo para calcular e mostrar o enésimo termo da série abaixo, onde o valor de (N) é lido do teclado.

5, 6, 11, 12, 17, 18, 23, 24, ...

23. Sendo $S = 1^2 + 2^2 + 3^2 +...+ n^2$, e "k", um número inteiro maior que 1, fazer um procedimento para calcular e o maior valor de "n" que torne a relação s < k verdadeira. O valor de "k" será lido do teclado.

24. O valor aproximado de π (PI) pode ser calculado usando a série:

$S = 1 - 1/3^3 + 1/5^3 - 1/7^3 + 1/9^3 \ldots$ e $\pi = \sqrt[3]{s \cdot 32}$

Fazer uma função para calcular o valor de π usando os 51 primeiros termos da série.

25. O número 3025 possui a seguinte característica:

30 + 25 = 55

$55^2 = 3025$

fazer um procedimento para mostrar todos os números de 4 algarismos que apresentem esta propriedade, a qual será verificada por intermédio de uma função.

26. O número 1221 possui a propriedade de que lido de "*trás-para-frente*" é igual ao lido de "*frente-para-trás*". Estes números são chamados de "palíndromos".
Calcular e mostrar todos os números palíndromos de 5 algarismos verificados por uma função.

27. Calcular e mostrar todos os números palíndromos menores que 30.000 e que sejam quadrados perfeitos, verificados por uma função.

28. Fazer um procedimento para calcular e mostrar os 100 primeiros termos da série de Fibonacci, esta série é gerada da seguinte forma: o primeiro e segundos termos valem 1 e os seguintes são calculados somando-se os dois termos anteriores a ele.

f = 1, 1, 2, 3, 5, 8, ...

29. Fazer um procedimento para calcular e mostrar os números primos compreendidos entre 500 e 600.

30. Fazer um algoritmo para :
- ler do teclado um conjunto de palavras, uma por vez;
- a última palavra será "FIM";
- contar o número de vezes em que as palavras "E", "OU" e "NÃO" aparecem neste conjunto, usando uma função para verificar a existência delas.