Capítulo 7 – Recursividade

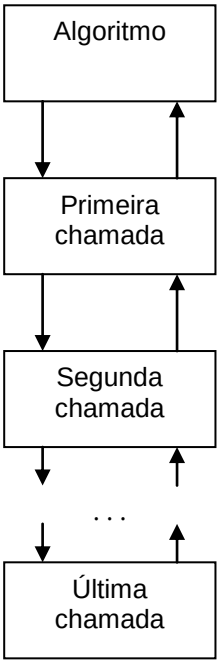
Introdução

Funções e procedimentos recursivos são aqueles que apresentam como característica poder chamar a si próprios.

Nestes casos, a cada nova chamada do procedimento, ou da função, as anteriores ficam suspensas até que esta última seja resolvida. Quando esta tiver sido, haverá um retorno sucessivo das chamadas suspensas, resolvendo-as uma a uma.

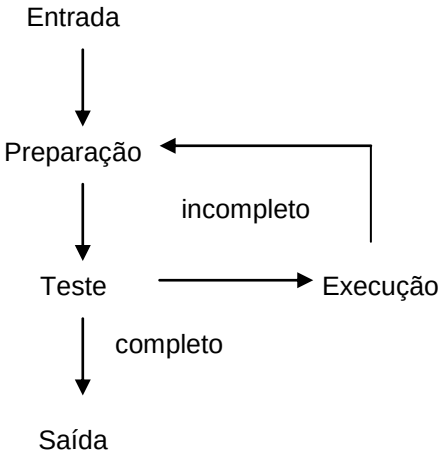
A utilização do conceito de recursividade pode simplificar bastante a elaboração de soluções para determinados tipos de problemas. Entretanto, de forma prática, às vezes, por limitações de espaço em memória, ou tempo, as soluções não recursivas sejam preferidas.

De forma geral, a recursividade pode ser representada como uma execução em profundidade.

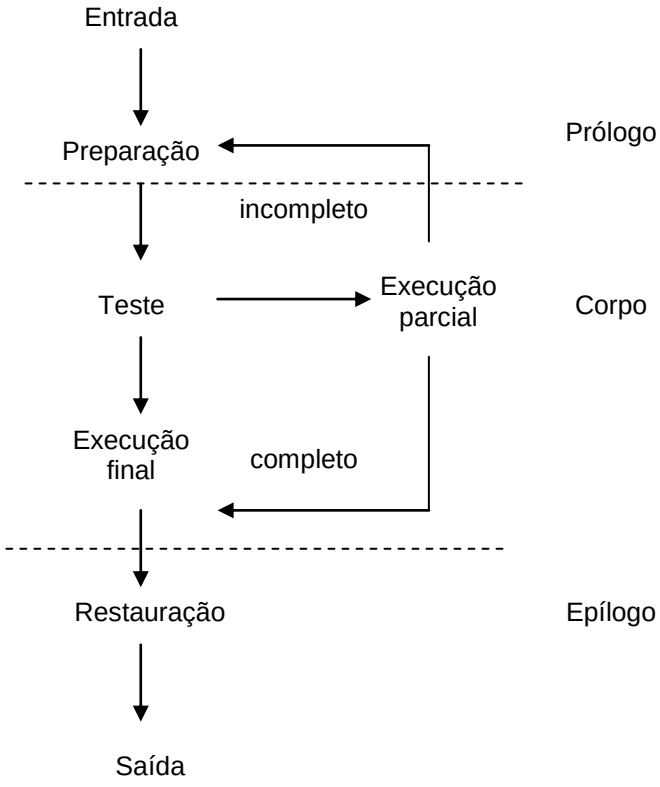


Os esquemas abaixo mostram as diferenças entre o funcionamento entre o emprego de iteração e de recursão.

Iteração :



Recursão :



Exemplo 1 :

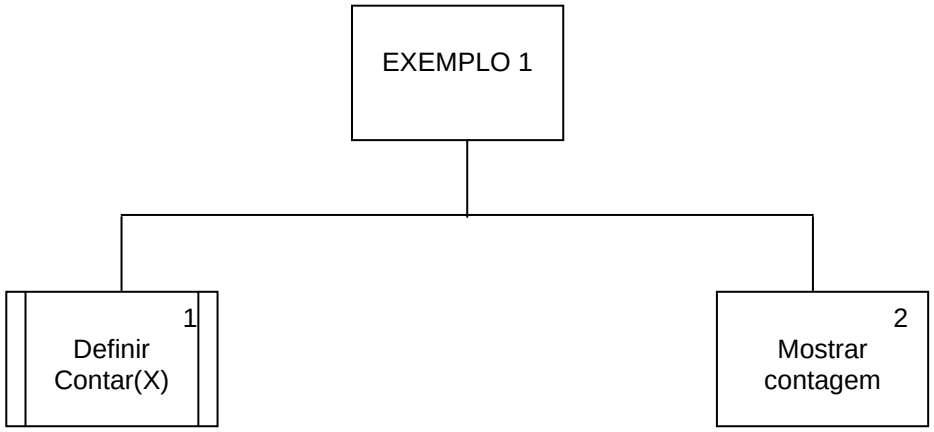
Fazer um programa com um procedimento para contar recursivamente de 1 até 10.

- Análise de dados:

- Dados do procedimento:

Dado	Tipo	Valor Inicial	Obs.
X	inteiro		valor no intervalo [1:10]

Diagrama funcional:



Algoritmo, com procedimento:

Esboço:

Primeira versão, só comentários.

Exemplo 1	v.1
Ação	Bloco
! definir procedimento CONTAR (inteiro X)	1
! parte principal	
! mostrar contagem de 1 até 10	2

Segunda versão, refinar o segundo bloco.

Exemplo 1	v.2
Ação	Bloco
! definir procedimento CONTAR (inteiro X)	1
! parte principal	
! mostrar contagem de 1 até 10 CONTAR (1); ! primeira chamada	2

Terceira versão, refinar o primeiro bloco.

Exemplo 1	v.3
Ação	Bloco
! definir procedimento	1
procedimento CONTAR (inteiro X)	
! verificar se o valor é válido	1,1
! mostrar valor	1.1.1
! passar ao próximo	1.2.1
! parte principal	
! mostrar contagem de 1 até 10 CONTAR (1); ! primeira chamada	2

Quarta versão, refinar novamente o primeiro bloco.

Exemplo 1			v.4
Ação			Bloco
! definir procedimento			1
procedimento CONTAR (inteiro X)			
! verificar se o valor é válido			1,1
$X \leq 10$?	V	! mostrar valor	1.1.1
		tela $\leftarrow$ (X, " ");	
		! passar ao próximo	1.2.1
		CONTAR (X + 1);	
! parte principal			
! mostrar contagem de 1 até 10 CONTAR (1); ! primeira chamada			2

Programa em SCILAB:

```
// Exemplo 1.
// Mostrar contagem de 1 ate' 10.
//
// definir funcao com parametro
function CONTAR ( X )
// parametro:
// int X;                          // numero de vezes
//
// 1.1 verificar se o valor e' valido
  if ( X <= 10 )
  // 1.1.1 mostrar valor
    printf ( "\n %d ", X );
  // 1.2.1 passar ao proximo
    CONTAR ( X + 1 );       // proxima chamada
  end // fim se
endfunction // CONTAR( )
//
//
// parte principal
//
  clc;          // limpar a area de trabalho
  CONTAR ( 1 ); // primeira chamada
// pausa para terminar
  printf  ( "\nPressionar qualquer tecla para terminar.\n" );
  halt;
// fim do programa
```

Programa em C++:

```
// Exemplo 1.
// Contar recursivamente de 1 ate 10.
//
// bibliotecas necessarias
#include <iostream>
using namespace std;
//                                                  // LINHAS DE PROGRAMA
// definir procedimento com parametro
void CONTAR ( int X )
{
 // 1.1 verificar se o valor e' valido
   if ( X <= 10 )                                   // # 1
   {
     // 1.1.1 mostrar valor
     cout << X << " ";                              // # 2
     // 1.2.1 passar ao proximo
     CONTAR ( X + 1 );        // proxima chamada    // # 3
   } // fim se
} // fim do procedimento CONTAR ( )                 // # 4
//
// parte principal
//
int main ( void )
{
// 2. mostrar contagem de 1ate' 10
   CONTAR ( 1 );              // primeira chamada   // # 5
// pausa para terminar
   cout << "\nPressionar qualquer tecla para terminar.";
   getchar ( );
   return EXIT_SUCCESS;
} // fim do programa
```

Execução :

Parâmetro : Linha :	I	tela	Observação
#5	1		primeira chamada
#1	X ≤ 10 ? V		
#2		1	
#3	2		segunda chamada
#1	X ≤ 10 ? V		
#2		2	
#3	3		terceira chamada
. . .			
#1	X ≤ 10 ? V		
#2		9	
#3	10		décima chamada
#1	X ≤ 10 ? V		
#2		10	
#3	11		última chamada
#1	X ≤ 10 ? F		
#4			retorno da última
#4			retorno da penúltima
#4			retorno à parte principal

Programa em C#:

```
/*
 * Exemplo 1
 * Contar recursivamente de 1 ate' 10.
 */
using System;

class Exemplo_1
{
// definir procedimento com parametro
   public static void CONTAR ( int X )
   {
   // 1.1 verificar se o valor e' valido
      if ( X <= 10 )
      {
      // 1.1.1 mostrar valor
         Console.Write ( "" + X + " " );
      // 1.2.1 passar ao proximo
         CONTAR ( X + 1 );     // proxima chamada
      } // fim se
   } // fim do procedimento CONTAR ( )
//
// parte principal
//
   public static void Main ( )
   {
   // 2. mostrar contagem de 1ate' 10
      CONTAR ( 1 );              // primeira chamada
   // pausa para terminar
      Console.Write ( "\nPressionar ENTER para terminar." );
      Console.ReadLine ( );
   } // end Main ( )
} // fim Exemplo_1 class
```

Programa em Java:

```
/**
  * Exemplo 1
  * Contar recursivamente de 1 ate' 10.
  */

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

class Exemplo_1
{
// definir procedimento com parametro
   public static void CONTAR ( int X )
   {
   // 1.1 verificar se o valor e' valido
      if ( X <= 10 )
      {
      // 1.1.1 mostrar valor
         IO.println ( "" + X + " " );
      // 1.2.1 passar ao proximo
         CONTAR ( X + 1 );     // proxima chamada
      } // fim se
   } // fim do procedimento CONTAR ( )
//
// parte principal
//
   public static void main ( String [ ] args )
   {
   // 2. mostrar contagem de 1ate' 10
      CONTAR ( 1 );              // primeira chamada
   // pausa para terminar
      IO.pause ( "\nPressionar ENTER para terminar." );
   } // end main ( )

} // fim Exemplo_1 class
```

Exemplo 2 :

Modificar o programa anterior para contar recursivamente de 10 até 1.

Esboço:

Primeira versão, só comentários.

Exemplo 2	v.1
Ação	Bloco
! definir procedimento CONTAR (inteiro X)	1
! parte principal	
! mostrar contagem de 10 até 1	2

Primeira versão, com refinamento do segundo bloco.

Exemplo 2	v.2
Ação	Bloco
! definir procedimento CONTAR (inteiro X)	1
! parte principal	
! mostrar contagem de 10 até 1 CONTAR (10); ! primeira chamada	2

Segunda versão, refinar o primeiro bloco.

Exemplo 2			v.3
Ação			Bloco
! definir procedimento			1
procedimento CONTAR (inteiro X)			
! verificar se o valor é válido			1,1
$X \leq 10$?	V	! passar ao próximo	1.1.1
		CONTAR (X + 1);	
		! mostrar valor	1.2.1
		tela ← (X, " ");	
! parte principal			
! mostrar contagem de 10 até 1 CONTAR (10); ! primeira chamada			2

Programa em SCILAB:

```
// Exemplo 2.
// Mostrar contagem de 10 ate' 1.
//
// definir funcao com parametro
function CONTAR ( X )
// parametro:
// int X;                              // numero de vezes
//
// 1.1 verificar se o valor e' valido
  if ( X <= 10 )
  // 1.1.1 passar ao próximo
    CONTAR ( X + 1 );                 // proxima chamada
  // 1.2.1 mostrar valor
    printf ( "\n %d ", X );
  end // fim se
endfunction // CONTAR( )
//
// parte principal
//
  clc;          // limpar a area de trabalho
  CONTAR ( 1 ); // primeira chamada
// pausa para terminar
  printf ( "\nPressionar qualquer tecla para terminar.\n" );
  halt;
// fim do programa
```

Programa em C++:

```
// Exemplo 2.
// Contar recursivamente de 10 ate' 1.
//
// bibliotecas necessarias
#include <conio.h>
#include <iostream.h>
//                                                  // LINHAS DE PROGRAMA
// definir procedimento com parametro
void CONTAR ( int X )
{
 // 1.1 verificar se o valor e' valido
   if ( X <= 10 )                                  // # 1
   {
     // 1.1.1 passar ao proximo
     CONTAR ( X + 1 );        // proxima chamada    // # 2
     // 1.2.1 mostrar valor
     cout << X << " ";                             // # 3
   } // fim se
} // fim do procedimento CONTAR ( )                 // # 4
//
// parte principal
//
int main ( void )
{
// 2. mostrar contagem de 10 ate' 1
   CONTAR ( 1 );              // primeira chamada   // # 5
// pausa para terminar
   cout << "\nPressionar qualquer tecla para terminar.";
   getchar ( );
   return EXIT_SUCCESS;
} // fim do programa
```

Execução :

Parâmetro : Linha :	I	tela	Observação
#5	1		primeira chamada
#1	X ≤ 10 ? V		
#2	2		segunda chamada
#1	X ≤ 10 ? V		
#2	3		terceira chamada
. . .			
#1	X ≤ 10 ? V		
#2	10		décima chamada
#1	X ≤ 10 ? V		
#2	11		última chamada
#1	X ≤ 10 ? F		
#4			retorno da última
#3		10	
#4			retorno da penúltima
#3		9	
#4			retorno da antepenúltima
. . .			
#3		2	
#4			retorno da segunda
#3		1	
#4			retorno à parte principal

Programa em C#:

```
/*
 * Exemplo 2
 * Contar recursivamente de 10 ate' 1.
 */
using System;

class Exemplo_2
{
// definir procedimento com parametro
   public static void CONTAR ( int X )
   {
   // 1.1 verificar se o valor e' valido
      if ( X <= 10 )
      {
      // 1.1.1 passar ao proximo
         CONTAR ( X + 1 );     // proxima chamada
      // 1.2.1 mostrar valor
         Console.Write ( "" + X + " " );
      } // fim se
   } // fim do procedimento CONTAR ( )
//
// parte principal
//
   public static void Main ( )
   {
   // 2. mostrar contagem de 1ate' 10
      CONTAR ( 1 );              // primeira chamada
   // pausa para terminar
      Console.Write ( "\nPressionar ENTER para terminar." );
      Console.ReadLine ( );
   } // end Main ( )


} // fim Exemplo_2 class
```

Programa em Java:

```
/**
  * Exemplo 2
  * Contar recursivamente de 10 ate' 1.
  */

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

class Exemplo_2
{
// definir procedimento com parametro
   public static void CONTAR ( int X )
   {
   // 1.1 verificar se o valor e' valido
      if ( X <= 10 )
      {
      // 1.1.1 passar ao proximo
         CONTAR ( X + 1 );     // proxima chamada
      // 1.2.1 mostrar valor
         IO.println ( "" + X + " " );
      } // fim se
   } // fim do procedimento CONTAR ( )
//
// parte principal
//
   public static void main ( String [ ] args )
   {
   // 2. mostrar contagem de 1ate' 10
      CONTAR ( 1 );              // primeira chamada
   // pausa para terminar
      IO.pause ( "\nPressionar ENTER para terminar." );
   } // end main ( )

} // fim Exemplo_2 class
```

Exemplo 3 :

Calcular o quociente inteiro da divisão de X por Y por uma função recursiva.

Análise do problema:

O quociente inteiro de um número (X) por outro (Y) pode ser calculado como uma contagem de quantas vezes é possível subtrair o segundo valor do primeiro.

Exemplo:

X = 5 e Y = 2

X > Y ? V => Q = 1 X = X – Y = 5 – 2 = 3

X > Y ? V => Q = 1+1 = 2 X = X – Y = 3 – 2 = 1

X > Y ? F => Q = 2 (quociente)

- Análise de dados:

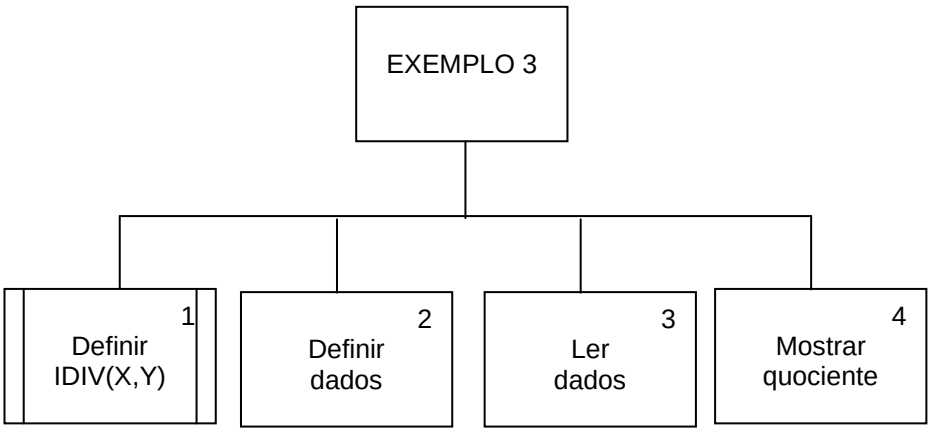
- Dados da função:

Dado	Tipo	Valor Inicial	Obs.
X	inteiro		dividendo
Y	inteiro		divisor
Q	inteiro		zero, se não puder dividir

- Dados da parte principal:

Dado	Tipo	Valor Inicial	Obs.
X	inteiro		dividendo
Y	inteiro		divisor

Diagrama funcional:



Algoritmo, com procedimento:

Esboço:

Primeira versão, só comentários.

Exemplo 3	v.1
Ação	Bloco
! definir função IDIV (inteiro X, inteiro Y)	1
! parte principal	
! definir dados locais	2
! ler dados	3
! calcular e mostrar o quociente	4

Segunda versão, com refinamento do segundo, terceiro e quarto blocos.

Exemplo 3	v.2
Ação	Bloco
! definir função IDIV (inteiro X, inteiro Y)	1
! parte principal	
! definir dados locais inteiro X, ! dividendo Y; ! divisor	2
! ler dados tela ← "Qual o valor de X ? "; X ← teclado; tela ← "Qual o valor de Y ? "; Y ← teclado;	3
! calcular e mostrar o quociente tela ← ("Quociente inteiro = ", IDIV (X,Y));	4

Terceira versão, refinar o primeiro bloco.

Exemplo 3			v.3
Ação			Bloco
! definir função			1
inteiro função IDIV (inteiro X, inteiro Y)			
! definir dado local inteiro Q; ! quociente			1.1
! verificar se pode subtrair			1.2
X > Y?	V	! contar e tentar fazer de novo Q = 1 + IDIV (X-Y, Y);	1.2.1
	F	Q = 0; ! não pode mais subtrair	1.2.2
retornar (Q);			
! parte principal			
! definir dados locais inteiro X, ! dividendo Y; ! divisor			2
! ler dados tela ← "Qual o valor de X ? "; X ← teclado; tela ← "Qual o valor de Y ? "; Y ← teclado;			3
! calcular e mostrar o quociente tela ← ("Quociente inteiro = ", IDIV (X,Y));			4

Programa em SCILAB:

```
// Exemplo 3.
// Calcular o quociente inteiro de X por Y
//
// definir funcao com parametros
function retorno = IDIV ( X, Y )
// parametros:
// int X;                    // dividendo
// int Y;                    // divisor
//
// 1.1 definir dado local
   Q = 0;                    // quociente
// 1.2 verificar se pode subtrair
   if ( X >= Y )
   // 1.2.1 contar e tentar de novo
      Q = 1 + IDIV ( X-Y, Y ); // proxima chamada
   else
   // 1.2.2 nao pode  mais subtrair
      Q = 0;
   end // fim se
// retornar o quociente
   retorno = Q;
endfunction // IDIV( )
//
//
// parte principal
//
// 2. definir dados locais
   X = 0;                    // dividendo
   Y = 0;                    // divisor
// 3. ler dados
   X = input ( "\nQual o valor de X ? " );
   Y = input ( "\nQual o valor de Y ? " );
// 4. calcular e mostrar o quociente
   printf ( "\nQuociente inteiro = %d ", IDIV ( X,Y ) );
// pausa para terminar
   printf ( "\nPressionar qualquer tecla para terminar.\n" );
   halt;
// fim do programa
```

Programa em C++:

```
// Exemplo 3.
// Calcular o quociente inteiro de X por Y recursivamente.
//
// bibliotecas necessárias
#include <iostream>
using namespace std;
//                                                  // LINHAS DE PROGRAMA
// definir funcao com parametros
int IDIV ( int X, int Y )
{
 // 1.1 definir dado local
   int Q;   // quociente
 // 1.2 verificar se pode subtrair
   if ( X >= Y )                                    // # 1
   {
     // 1.2.1 contar e tentar de novo
     Q = 1 + IDIV ( X-Y, Y );  // próxima chamada    // # 2
   }
   else
   {
     // 1.2.2 nao pode mais subtrair
     Q = 0;                                         // # 3
   } // fim se
   return ( Q );                                    // # 4
} // fim da funcao IDIV ( )
//
// parte principal
//
int main ( void )
{
// 2. definir dados locais
   int X,                 // dividendo
       Y;                 // divisor
// 3. ler dados
   cout << "\nQual o valor de X ? ";   cin >> X;          // #5
   cout << "\nQual o valor de Y ? ";   cin >> Y;          // #6
// 4. calcular e mostrar o quociente
   cout << "\nQuociente inteiro = " << IDIV ( X,Y );     // #7
// pausa para terminar
   cout << "\nPressionar qualquer tecla para terminar.";
   getchar ( );
   return EXIT_SUCCESS;
} // fim do programa
```

Execução :

Dados : X = 4 e Y = 2

Parâmetros: Linha :	X	Y	IDIV	teclado	tela	Observação
#5	-	-		5		
#6	5	-		2		
#7	5	2	-			primeira chamada
#1	X ≥ Y ? V					
#2	3	2	1+IDIV(5-2,2)			segunda chamada
#1	X ≥ Y ? V					
#2	1	2	1+IDIV(3-2,2)			terceira chamada
#1	X ≥ Y ? F					
#3	0	2				
#4			0			primeiro retorno
#3			1+0			
#4			1			segundo retorno
#3			1+1			
#4			2			terceiro retorno
#6	5	2			2	parte principal

Programa em C#:

```
/*
 * Exemplo 3
 * Calcular o quociente inteiro de X por Y recursivamente.
 */
using System;

class Exemplo_3
{
// definir funcao com parametros
   public static int IDIV ( int X, int Y )
   {
   // 1.1. definir dado local
      int Q;               // quociente
   // 1.2. verificar se pode subtrair
      if ( X >= Y )
      {
      // 1.2.1 contar e tentar de novo
         Q = 1 + IDIV ( X-Y, Y );          // proxima chamada
      }
      else
      {
      // 1.2.2 nao pode mais subtrair
         Q = 0;
      } // fim se
      return ( Q );
   } // fim da funcao IDIV ( )
//
// parte principal
//
   public static void Main ( )
   {
   // 2. definir dados locais
      int X,               // dividendo
          Y;               // divisor
   // 3. ler dados
      Console.Write ( "\nQual o valor de X ?" );
      X = int.Parse ( Console.ReadLine ( ) );
      Console.Write ( "\nQual o valor de Y ?" );
      Y = int.Parse ( Console.ReadLine ( ) );
   // 4. calcular e mostrar o quociente
      Console.WriteLine ( "\nQuociente inteiro = " + IDIV ( X, Y ) );
   // pausa para terminar
      Console.Write ( "\nPressionar ENTER para terminar." );
      Console.ReadLine ( );
   } // end Main ( )

} // fim Exemplo_3 class
```

Programa em Java:

```
/**
  * Exemplo 3
  * Calcular o quociente inteiro de X por Y recursivamente.
  */

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

class Exemplo_3
{
// definir funcao com parametros
   public static int IDIV ( int X, int Y )
   {
   // 1.1. definir dado local
      int Q;               // quociente
   // 1.2. verificar se pode subtrair
      if ( X >= Y )
      {
      // 1.2.1 contar e tentar de novo
         Q = 1 + IDIV ( X-Y, Y );          // proxima chamada
      }
      else
      {
      // 1.2.2 nao pode mais subtrair
         Q = 0;
      } // fim se
      return ( Q );
   } // fim da funcao IDIV ( )
//
// parte principal
//
   public static void main ( String [ ] args )
   {
   // 2. definir dados locais
      int X,               // dividendo
          Y;               // divisor
   // 3. ler dados
      X = IO.readint ( "\nQual o valor de X ? " );
      Y = IO.readint ( "\nQual o valor de Y ? " );
   // 4. calcular e mostrar o quociente
      IO.println ( "\nQuociente inteiro = " + IDIV ( X, Y ) );
   // pausa para terminar
      IO.pause ( "\nPressionar ENTER para terminar." );
   } // end main ( )

} // fim Exemplo_3 class
```

Exemplo 4 :

Calcular o resto inteiro da divisão de X por Y por uma função recursiva.

Análise do problema:

O resto inteiro de um número (X) por outro (Y) pode ser calculado como o que sobra após sucessivas subtrações do segundo valor em relação ao primeiro.

Exemplo:

X = 5 e Y = 2

X > Y ? V => R = 3 X = X – Y = 5 – 2 = 3

X > Y ? V => R = 1 X = X – Y = 3 – 2 = 1

X > Y ? F => R = 1 (resto)

- Análise de dados:

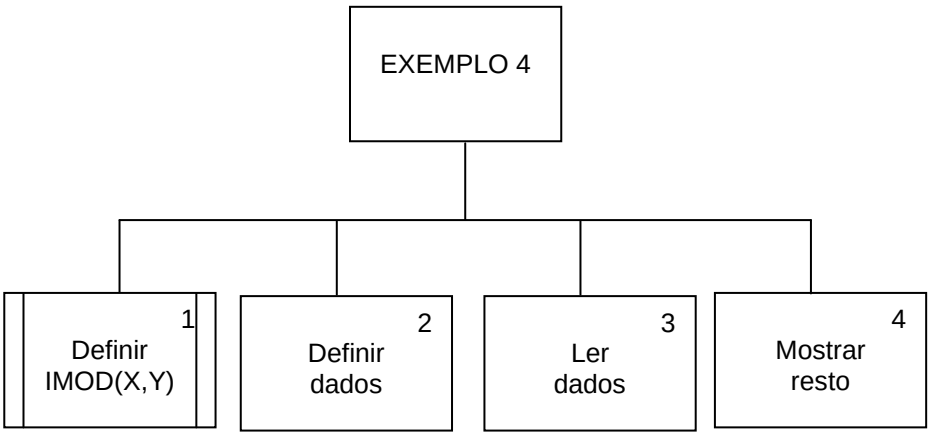
- Dados da função:

Dado	Tipo	Valor Inicial	Obs.
X	inteiro		dividendo
Y	inteiro		divisor
R	inteiro		resto

- Dados da parte principal:

Dado	Tipo	Valor Inicial	Obs.
X	inteiro		dividendo
Y	inteiro		divisor

Diagrama funcional:



Algoritmo, com função:

Esboço:

Primeira versão, só comentários.

Exemplo 4	v.1
Ação	Bloco
! definir função IMOD (inteiro X, inteiro Y)	1
! parte principal	
! definir dados locais	2
! ler dados	3
! calcular e mostrar o resto	4

Segunda versão, com refinamento do segundo, terceiro e quarto blocos.

Exemplo 4	v.2
Ação	Bloco
! definir função IMOD (inteiro X, inteiro Y)	1
! parte principal	
! definir dados locais inteiro X, ! dividendo Y; ! divisor	2
! ler dados tela ← "Qual o valor de X ? "; X ← teclado; tela ← "Qual o valor de Y ? "; Y ← teclado;	3
! calcular e mostrar o resto tela ← ("Resto inteiro = ", IMOD (X,Y));	4

Terceira versão, refinar o primeiro bloco.

Exemplo 4			v.3
Ação			Bloco
! definir função			1
inteiro IMOD (inteiro X, inteiro Y)			
! definir dado local inteiro R; ! resto			1.1
! verificar se pode subtrair			1.2
X>=Y?	V	! contar e tentar fazer de novo R = IMOD (X-Y, Y);	1.2.1
	F	R = X; ! não pode mais subtrair	1.2.2
retornar (R);			
! parte principal			
! definir dados locais inteiro X, ! dividendo Y; ! divisor			2
! ler dados tela ← "Qual o valor de X ? "; X ← teclado; tela ← "Qual o valor de Y ? "; Y ← teclado;			3
! calcular e mostrar o resto inteiro tela ← ("Resto inteiro = ", IMOD (X,Y));			4

Programa em SCILAB:

```
// Exemplo 4.
// Calcular o resto inteiro de X por Y recursivamente
//
// definir funcao com parametros
function retorno = IMOD ( X, Y )
// parametros:
// int X;                    // dividendo
// int Y;                    // divisor
//
// 1.1 definir dado local
   R = 0;                    // resto
// 1.2 verificar se pode subtrair
   if ( X >= Y )
   // 1.2.1 tentar de novo
      R = IMOD ( X-Y, Y );   // proxima chamada
   else
   // 1.2.2 nao pode  mais subtrair
      R = X;
   end // fim se
// retornar o resto
   retorno = R;
endfunction // IMOD( )
//
// parte principal
//
// 2. definir dados locais
   X = 0;                    // dividendo
   Y = 0;                    // divisor
// 3. ler dados
   clc;    // limpar a area de trabalho
   X = input ( "\nQual o valor de X ? " );
   Y = input ( "\nQual o valor de Y ? " );
// 4. calcular e mostrar o quociente
   printf ( "\nResto inteiro = %d ", IMOD ( X, Y ) );
// pausa para terminar
   printf ( "\nPressionar qualquer tecla para terminar.\n" );
   halt;
// fim do programa
```

Programa em C++:

```
// Exemplo 4.
// Calcular o resto inteiro de X por Y recursivamente.
//
// bibliotecas necessarias
#include <iostream>
using namespace std;
//                                                    // LINHAS DE PROGRAMA
// definir funcao com parâmetros
int IMOD ( int X, int Y )
{
 // 1.1 definir dado local
   int R;   // resto
 // 1.2 verificar se pode subtrair
   if ( X >= Y )                                     // # 1
   {
     // 1.2.1 tentar de novo
     R = IMOD ( X-Y, Y );      // próxima chamada     // # 2
   }
   else
   {
     // 1.2.2 nao pode mais subtrair
     R = X;                                          // # 3
   } // fim se
   return ( R );                                     // # 4
} // fim da funcao IMOD ( )
//
// parte principal
//
int main ( void )
{
// 2. definir dados locais
   int X,                    // dividendo
       Y;                    // divisor
// 3. ler dados
   cout << "\nQual o valor de X ? ";   cin >> X;     // #5
   cout << "\nQual o valor de Y ? ";   cin >> Y;     // #6
// 4. calcular e mostrar o resto
   cout << "\nResto inteiro = " << IMOD ( X,Y );     // #7
// pausa para terminar
   cout << "\nPressionar qualquer tecla para terminar.";
   getchar ( );
   return EXIT_SUCCESS;
} // fim do programa
```

Execução :

Dados : X = 5 e Y = 2

Parâmetros: Linha :	X	Y	IMOD	teclado	tela	Observação
#5	-	-		5		
#6	5	-		2		
#7	5	2	-			primeira chamada
#1	X ≥ Y ? V					
#2	5	2	IMOD(5-2,2)			segunda chamada
#1	X ≥ Y ? V					
#2	3	2	IMOD(3-2,2)			terceira chamada
#1	X ≥ Y ? F					
#3	1	2				
#4			1			primeiro retorno
#3	3	2				
#4			1			segundo retorno
#3	5	2				
#4			1			terceiro retorno
#7	5	2	1		1	parte principal

Dados : X = 4 e Y = 2

Parâmetros: Linha :	X	Y	IMOD	teclado	tela	Observação
#5	-	-		5		
#6	4	-		2		
#7	4	2	-			primeira chamada
#1	X ≥ Y ? V					
#2	4	2	IMOD(4-2,2)			segunda chamada
#1	X ≥ Y ? V					
#2	2	2	IMOD(2-2,2)			terceira chamada
#1	X ≥ Y ? F					
#3	0	2				
#4			0			primeiro retorno
#3	2	2				
#4			0			segundo retorno
#3	4	2				
#4			0			terceiro retorno
#7	4	2	0		0	parte principal

Programa em C#:

```
/*
 * Exemplo 4
 * Calcular o resto inteiro da divisao de X por Y recursivamente.
 */
using System;

class Exemplo_4
{
// definir funcao com parametros
   public static int IMOD ( int X, int Y )
   {
   // 1.1. definir dado local
      int R               // resto
   // 1.2. verificar se pode subtrair
      if ( X >= Y )
      {
      // 1.2.1 contar e tentar de novo
         R = IMOD ( X-Y, Y );  // proxima chamada
      }
      else
      {
      // 1.2.2 nao pode mais subtrair
         R = X;
      } // fim se
      return ( R );
   } // fim da funcao IMOD ( )
//
// parte principal
//
   public static void Main ( )
   {
   // 2. definir dados locais
      int X,              // dividendo
          Y;              // divisor
   // 3. ler dados
      Console.Write ( "\nQual o valor de X ?" );
      X = int.Parse ( Console.ReadLine ( ) );
      Console.Write ( "\nQual o valor de Y ?" );
      Y = int.Parse ( Console.ReadLine ( ) );
   // 4. calcular e mostrar o resto
      Console.WriteLine ( "\nResto inteiro = " + IMOD ( X, Y ) );
   // pausa para terminar
      Console.Write ( "\nPressionar ENTER para terminar." );
      Console.ReadLine ( );
   } // end Main ( )

} // fim Exemplo_4 class
```

Programa em Java:

```
/**
  * Exemplo 4
  * Calcular o resto inteiro da divisao de X por Y recursivamente.
  */

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

class Exemplo_4
{
// definir funcao com parametros
   public static int IMOD ( int X, int Y )
   {
   // 1.1. definir dado local
      int R                          // resto
   // 1.2. verificar se pode subtrair
      if ( X >= Y )
      {
      // 1.2.1 contar e tentar de novo
         R = IMOD ( X-Y, Y );  // proxima chamada
      }
      else
      {
      // 1.2.2 nao pode mais subtrair
         R = X;
      } // fim se
      return ( R );
   } // fim da funcao IMOD ( )
//
// parte principal
//
   public static void main ( String [ ] args )
   {
   // 2. definir dados locais
      int X,                         // dividendo
          Y;                         // divisor
   // 3. ler dados
      X = IO.readint ( "\nQual o valor de X ? " );
      Y = IO.readint ( "\nQual o valor de Y ? " );
   // 4. calcular e mostrar o resto
      IO.println ( "\nResto inteiro = " + IMOD ( X, Y ) );
   // pausa para terminar
      IO.pause ( "\nPressionar ENTER para terminar." );
   } // end main ( )

} // fim Exemplo_4 class
```

Exemplo 5 :

Calcular o máximo divisor comum entre X e Y por uma função recursiva.

Análise do problema:

O máximo divisor comum pode ser definido recursivamente como:

$$MDC(X,Y) = \begin{cases} X, & \text{se } Y = 0 \\ MDC(Y, X//Y), & \text{se } Y \neq 0 \end{cases}$$

Exemplo:

X = 12 e Y = 8

Y == 0 ? F => MDC (12, 8) = MDC (8, 12//8) = MDC (8, 4)
Y == 0 ? F => MDC (8, 4) = MDC (4, 8//4) = MDC (4, 0)
Y == 0 ? V => MDC 4

- Análise de dados:

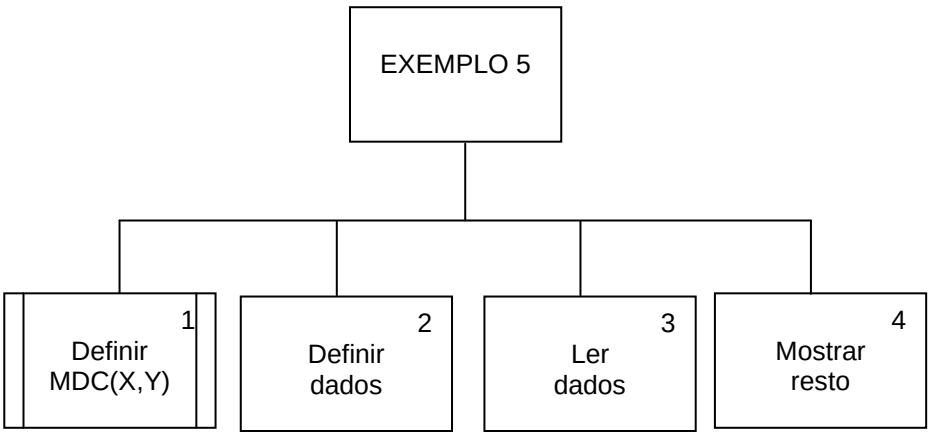
- Dados da função:

Dado	Tipo	Valor Inicial	Obs.
X	inteiro		dividendo
Y	inteiro		divisor
Resultado	inteiro		máximo divisor comum

- Dados da parte principal:

Dado	Tipo	Valor Inicial	Obs.
X	inteiro		dividendo
Y	inteiro		divisor

Diagrama funcional:



Algoritmo, com função:

Esboço:

Primeira versão, só comentários.

Exemplo 5	v.1
Ação	Bloco
! definir função MDC (inteiro X, inteiro Y)	1
! parte principal	
! definir dados locais	2
! ler dados	3
! calcular e mostrar o quociente	4

Segunda versão, com refinamento do segundo, terceiro e quarto blocos.

Exemplo 4	v.2
Ação	Bloco
! definir função MDC (inteiro X, inteiro Y)	1
! parte principal	
! definir dados locais inteiro X, ! dividendo Y; ! divisor	2
! ler dados tela ← "Qual o valor de X ? "; X ← teclado; tela ← "Qual o valor de Y ? "; Y ← teclado;	3
! calcular e mostrar o M.D.C tela ← ("M.D.C. = ", MDC (X,Y));	4

Terceira versão, refinar o primeiro bloco.

Exemplo 5			v.3
Ação			Bloco
! definir função			1
inteiro MDC (inteiro X, inteiro Y)			
! definir dado local inteiro Resultado;			1.1
! verificar se pode subtrair			1.2
Y==0?	V	Resultado = X;	1.2.1
	F	Resultado = MDC (Y, X//Y);	1.2.2
retornar (Resultado);			
! parte principal			
! definir dados locais inteiro X, ! dividendo Y; ! divisor			2
! ler dados tela ← "Qual o valor de X ? "; X ← teclado; tela ← "Qual o valor de Y ? "; Y ← teclado;			3
! calcular e mostrar o M.D.C tela ← ("M.D.C. = ", MDC (X,Y));			4

Programa em C++:

```
// Exemplo 5.
// Calcular o maximo divisor comum de X e Y recursivamente.
//
// bibliotecas necessarias
#include <iostream>
using namespace std;
//                                                    // LINHAS DE PROGRAMA
// definir de funcao com parametros
int MDC ( int X, int Y )
{
 // 1.1 definir dado local
   int Resultado;              // maximo divisor comum
 // 1.2 verificar se pode calcular
   if ( Y == 0 )                                     // # 1
   {
     // 1.2.1 nao pode mais calcular
      Resultado = X;                                 // # 2
   }
   else
   {
    // 1.2.2 inverter os dados e continuar
      Resultado = MDC ( Y, X//Y );                   // # 3
   } // fim se
   return ( Resultado );                             // # 4
} // fim da funcao MDC ( )
//
// parte principal
//
int main ( void )
{
// 2. definir dados locais
   int X,                      // dividendo
       Y;                      // divisor
// 3. ler dados
   cout << "\nQual o valor de X ? ";   cin >> X;     // #5
   cout << "\nQual o valor de Y ? ";   cin >> Y;     // #6
// 4. calcular e mostrar o maximo divisor comum
   cout << "\nM.D.C. = " << MDC ( X, Y );           // #7
// pausa para terminar
   cout << "\nPressionar qualquer tecla para terminar.";
   getchar ( );
   return EXIT_SUCCESS;
} // fim do programa
```

Execução :

Dados : X = 12 e Y = 8

Parâmetros: Linha :	X	Y	MDC Resultado	teclado	tela	Observação
#5	-	-		12		
#6	12	-		8		
#7	12	8	MDC(12, 8)			primeira chamada
#1		Y = 0 ? F				
#3	12	8	MDC(8,12//8)			segunda chamada
#1		Y = 0 ? F				
#3	8	4	MDC(4,8//4)			terceira chamada
#1		Y = 0 ? V				
#2	4	0	MDC(4, 0)			quarta chamada
#4	4	0	4			primeiro retorno
#3	8	4	4			
#4	8	4	4			segundo retorno
#3	12	8	4			
#4	12	8	4			terceiro retorno
#7	12	8	4		4	parte principal

Programa em C#:

```
/**
  * Exemplo 5
  * Calcular o maximo divisor comum de X por Y recursivamente.
  */
using System;

class Exemplo_5
{
// definir funcao com parametros
   public static int MDC ( int X, int Y )
   {
   // 1.1. definir dado local
      int Resultado;    // maximo divisor comum
   // 1.2. verificar se pode subtrair
      if ( Y == 0 )
      {
      // 1.2.1 nao pode mais calcular
         Resultado = X;
      }
      else
      {
      // 1.2.2 inverter os dados e continuar
         Resultado = MDC ( Y, X//Y );
      } // fim se
      return ( Resultado );
   } // fim da funcao MDC ( )
//
// parte principal
//
   public static void Main ( )
   {
   // 2. definir dados locais
      int X,               // dividendo
          Y;               // divisor
   // 3. ler dados
      Console.Write ( "\nQual o valor de X ?" );
      X = int.Parse ( Console.ReadLine ( ) );
      Console.Write ( "\nQual o valor de Y ?" );
      Y = int.Parse ( Console.ReadLine ( ) );
   // 4. calcular e mostrar o resto
      Console.WriteLine ( "\nM.D.C. = " + MDC ( X, Y ) );
   // pausa para terminar
      Console.Write ( "\nPressionar ENTER para terminar." );
      Console.ReadLine ( );
   } // end Main ( )

} // fim Exemplo_5 class
```

Programa em Java:

```
/**
  * Exemplo 5
  * Calcular o maximo divisor comum de X por Y recursivamente.
  */

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

class Exemplo_5
{
// definir funcao com parametros
   public static int MDC ( int X, int Y )
   {
   // 1.1. definir dado local
      int Resultado;              // maximo divisor comum
   // 1.2. verificar se pode subtrair
      if ( Y == 0 )
      {
      // 1.2.1 nao pode mais calcular
         Resultado = X;
      }
      else
      {
      // 1.2.2 inverter os dados e continuar
         Resultado = MDC ( Y, X/Y );
      } // fim se
      return ( Resultado );
   } // fim da funcao MDC ( )
//
// parte principal
//
   public static void main ( String [ ] args )
   {
   // 2. definir dados locais
      int X,                        // dividendo
          Y;                        // divisor
   // 3. ler dados
      X = IO.readint ( "\nQual o valor de X ? " );
      Y = IO.readint ( "\nQual o valor de Y ? " );
   // 4. calcular e mostrar o resto
      IO.println ( "\nM.D.C. = " + MDC ( X, Y ) );
   // pausa para terminar
      IO.pause ( "\nPressionar ENTER para terminar." );
   } // end main ( )

} // fim Exemplo_5 class
```

Exemplo 6 :

Calcular a soma dos 50 primeiros números naturais por uma função recursiva.

$$S(N) = 1 + 2 + 3 + ... + 49 + 50$$

Análise do problema:

O somatório pode ser definido recursivamente como:

$$S(X) = \begin{cases} 0, & \text{se } X \leq 0 \\ X + S(X + 1), & \text{se } X \in [1:50] \end{cases}$$

- Análise de dados:

- Dados da função:

Dado	Tipo	Valor Inicial	Obs.
X	inteiro		valor da vez
S	real		somatório

- Avaliação da solução :

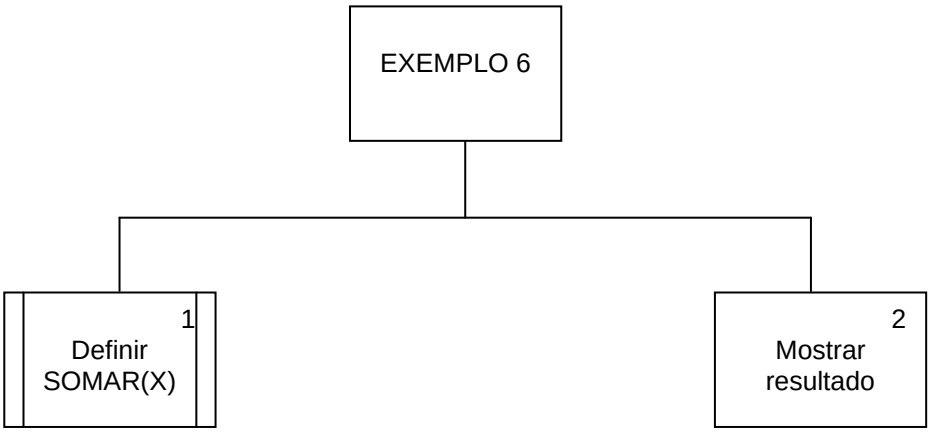
Calculando pela fórmula :

$$\sum_{i-1}^{n} i = \frac{n(n+1)}{2} \Rightarrow \sum_{i-1}^{50} i = \frac{50(50+1)}{2} = 1275$$

Resultado :

SOMA = 1275

Diagrama funcional:



Algoritmo, com função:

Esboço:

Primeira versão, só comentários.

Exemplo 6	v.1
Ação	Bloco
! definir função SOMAR (inteiro X)	1
! parte principal	
! calcular e mostrar o somatório	2

Segunda versão, com refinamento do segundo bloco.

Exemplo 6	v.2
Ação	Bloco
! definir função SOMAR (inteiro X)	1
! parte principal	
! calcular e mostrar o somatório tela ← ("\nS = ", SOMAR (1));	2

Terceira versão, refinar o primeiro bloco.

Exemplo 6			v.3
Ação			Bloco
! definir função			1
inteiro SOMAR (inteiro X)			
! definir dado local inteiro S;			1.1
! verificar se X está fora do limite			1.2
X>50?	V	S = 0;	1.2.1
	F	S= X + SOMAR(X+1);	1.2.2
retornar (S);			
! parte principal			
! calcular e mostrar o somatório tela ← ("S = ", SOMAR (1));			2

Programa em SCILAB:

```
// Exemplo 6.
// Calcular o somatorio dos 50 primeiros numeros naturais, recursivamente.
//
// definir funcao com parametro
function retorno = SOMAR ( X )
// parametro:
// int X;                    // natural
//
// 1.1 definir dado local
   S = 0;                    // somatorio
// 1.2 verificar se X esta' fora do limite
   if ( X > 50 )
   // 1.2.1 nao ha' mais o que somar
      S = 0;                 // comecar a retornar
   else
   // 1.2.2 somar se o termo for valido
      S = X + SOMAR ( X+1 );// proxima chamada
   end // fim se
// retornar o resultado
   retorno = S;
endfunction // SOMAR( )
//
// parte principal
//
// 2. calcular e mostrar o somatorio
   clc;   // limpar a area de trabalho
   printf ( "\nS(N) = %d ", SOMAR ( 1 ) );
// pausa para terminar
   printf ( "\nPressionar qualquer tecla para terminar.\n" );
   halt;
// fim do programa
```

Programa em C++:

```
// Exemplo 6.
// Calcular o somatorio dos 50 primeiros numeros naturais, recursivamente.
//
// bibliotecas necessarias
#include <iostream>
using namespace std;

//                                                    // LINHAS DE PROGRAMA
// definir funcao com parametro
int SOMAR ( int X )
{
 // 1.1 definir dado local
   int S;                          // somatorio
 // 1.2 verificar se X esta' fora do limite
   if ( X > 50 )                                     // # 1
   {
     // 1.2.1 nao ha' mais o que somar
     S = 0;                        // comecar a retornar   // # 2
   }
   else
   {
     // 1.2.2 somar se for termo valido
      S = X + SOMAR ( X+1 );                         // # 3
   } // fim se
   return ( S );                                     // # 4
} // fim da funcao SOMAR ( )
//
// parte principal
//
int main ( void )
{
// 2. calcular e mostrar o somatorio
   cout << "\nS(N) = " << SOMAR ( 1 );               // #5
// pausa para terminar
   cout << "\nPressionar qualquer tecla para terminar.";
   getchar ( );
   return EXIT_SUCCESS;
} // fim do programa
```

Execução :

Parâmetros:	X		SOMAR	teclado	tela	Observação
Linha :			S			
#5	-					
#6			SOMAR(1)			primeira chamada
#1	1	X>50 ? F	0			
#3			1+SOMAR(2)			segunda chamada
#1	2	X>50 ? F	0			
#3			2+SOMAR(3)			terceira chamada
#1	3	X>50 ? F	0			
#3			3+SOMAR(4)			quarta chamada
			0			
			. . .			
#1	50	X>50 ? F				
#3			50+SOMAR(51)			última chamada
#1	51	X>50 ? V				
#2			0			
#4			0			primeiro retorno
#3	50		50 + 0			
#4			50			segundo retorno
#3	49		49 + 50			
			. . .			
#4			(3+4+...+49+50)			penúltimo retorno
#3	2		2+(3+...+50)			
#4			(2+3+...+50)			último retorno
#3	1		1+(2+...+50)			
#4			(1+...+50)			
#5			(1 + ... + 50)			parte principal

Programa em C#:

```
/*
 * Exemplo 6
 * Calcular o somatorio dos 50 primeiros numeros naturais, recursivamente.
 */
using System;

class Exemplo_6
{
// definir funcao com parametro
   public static int SOMAR ( int X )
   {
   // 1.1. definir dado local
      int S;               // somatorio
   // 1.2. verificar se X esta' fora do limite
      if ( X > 50 )
      {
      // 1.2.1 nao ha' mais o que somar
         S = 0;
      }
      else
      {
      // 1.2.2 somar se for termo valido
         S = X + SOMAR ( X+1 );
      } // fim se
      return ( S );
   } // fim da funcao SOMAR ( )
//
// parte principal
//
   public static void Main ( )
   {
   // 2. calcular e mostrar o somatorio
      Console.WriteLine ( "\nS(N) = " + SOMAR ( 1 ) );
   // pausa para terminar
      Console.Write ( "\nPressionar ENTER para terminar." );
      Console.ReadLine ( );
   } // end Main ( )

} // fim Exemplo_6 class
```

Programa em Java:

```
/**
  * Exemplo 6
  * Calcular o somatorio dos 50 primeiros numeros naturais, recursivamente.
  */

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

class Exemplo_6
{
// definir funcao com parametro
   public static int SOMAR ( int X )
   {
   // 1.1. definir dado local
      int S;                              // somatorio
   // 1.2. verificar se X esta' fora do limite
      if ( X > 50 )
      {
      // 1.2.1 nao ha' mais o que somar
         S = 0;
      }
      else
      {
      // 1.2.2 somar se for termo valido
         S = X + SOMAR ( X+1 );
      } // fim se
      return ( S );
   } // fim da funcao SOMAR ( )
//
// parte principal
//
   public static void main ( String [ ] args )
   {
   // 2. calcular e mostrar o somatorio
      IO.println ( "\nS(N) = " + SOMAR ( 1 ) );
   // pausa para terminar
      IO.pause ( "\nPressionar ENTER para terminar." );
   } // end main ( )

} // fim Exemplo_6 class
```

Exemplo 7 :

Calcular a soma dos 10 primeiros números pares por uma função recursiva.

S(N) = 2 + 4 + ... + 18 + 20

Análise do problema:

O somatório pode ser definido recursivamente como:

$$S(X) = \begin{cases} 0, & \text{se } X \leq 0 \\ 2*X + S(X - 1), & \text{se } X \in [1:10] \end{cases}$$

- Análise de dados:

- Dados da função:

Dado	Tipo	Valor Inicial	Obs.
X	inteiro		valor da vez
S	real		somatório

- Avaliação da solução :

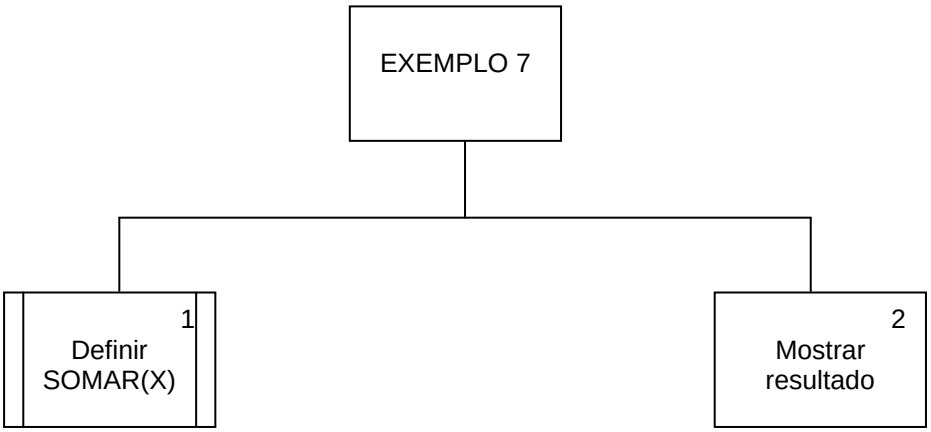
Calculando pela fórmula :

$$\sum_{i-1}^{n} 2i = 2\sum_{i-1}^{n} i = 2\frac{n(n+1)}{2} = \ n(n+1) \Rightarrow \sum_{i-1}^{10} 2i = [10(10+1)] = 110$$

Resultado :

SOMA = 110

Diagrama funcional:



Algoritmo, com função:

Esboço:

Primeira versão, só comentários.

Exemplo 7	v.1
Ação	Bloco
! definir função SOMAR (inteiro X)	1
! parte principal	
! calcular e mostrar o somatório	2

Segunda versão, com refinamento do segundo bloco.

Exemplo 7	v.2
Ação	Bloco
! definir função SOMAR (inteiro X)	1
! parte principal	
! calcular e mostrar o somatório tela ← ("\nS = ", SOMAR (10));	2

Terceira versão, refinar o primeiro bloco.

Exemplo 7			v.3
Ação			Bloco
! definir função			1
inteiro SOMAR (inteiro X)			
! definir dado local inteiro S;			1.1
! verificar se X está fora do limite			1.2
X==0?	V	S = 0;	1.2.1
	F	S= 2*X + SOMAR(X-1);	1.2.2
retornar (S);			
! parte principal			
! calcular e mostrar o somatório tela ← ("\nS = ", SOMAR (10));			2

Programa em SCILAB:

```
// Exemplo 7.
// Calcular o somatorio dos 10 primeiros pares, recursivamente
//
// definir funcao com parametro
function retorno = SOMAR ( X )
// parametro:
// int X;                       // natural
//
// 1.1 definir dado local
   S = 0;                       // somatorio
// 1.2 verificar se X esta' fora do limite
   if ( X == 0 )
   // 1.2.1 nao ha' mais o que somar
      S = 0;                    // comecar a retornar
   else
   // 1.2.2 somar se o termo for valido
      S = 2*X + SOMAR ( X-1 );      // proxima chamada
   end // fim se
// retornar o resultado
   retorno = S;
endfunction // SOMAR( )
//
// parte principal
//
// 2. calcular e mostrar o somatorio
   clc;    // limpar a area de trabalho
   printf ( "\nS(N) = %d ", SOMAR ( 10 ) );
// pausa para terminar
   printf ( "\nPressionar qualquer tecla para terminar.\n" );
   halt;
// fim do programa
```

Programa em C++:

```
// Exemplo 7.
// Calcular o somatorio dos 10 primeiros pares, recursivamente.
//
// bibliotecas necessárias
#include <iostream>
using namespace std;
//                                                  // LINHAS DE PROGRAMA
// definir funcao com parametro
int SOMAR ( int X )
{
 // 1.1 definir dado local
   int S;                          // somatorio
 // 1.2 verificar se X esta' fora do limite
   if ( X == 0 )                                    // # 1
   {
     // 1.2.1 nao ha' mais o que somar
     S = 0;                        // comecar a retornar    // # 2
   }
   else
   {
     // 1.2.2 somar se for termo valido
      S = 2*X + SOMAR ( X-1 );                      // # 3
   } // fim se
   return ( S );                                    // # 4
} // fim da funcao SOMAR ( )
//
// parte principal
//
int main ( void )
{
// 2. calcular e mostrar o somatorio
   cout << "\nS(N) = " << SOMAR ( 10 );             // #5
// pausa para terminar
   cout << "\nPressionar qualquer tecla para terminar.";
   getchar ( );
   return EXIT_SUCCESS;
} // fim do programa
```

Execução :

Parâmetros:	X		SOMAR	teclado	tela	Observação
Linha :			S			
#5	-					
#6			SOMAR(10)			primeira chamada
#1	10	X==0 ? F	0			
#3			20+SOMAR(9)			segunda chamada
#1	9	X==0 ? F	0			
#3			18+SOMAR(8)			terceira chamada
#1	8	X==0 ? F	0			
#3			16+SOMAR(7)			quarta chamada
			0			
			. . .			
#1	1	X==0 ? F				
#3			2+SOMAR(0)			última chamada
#1	0	X==0 ? F				
#2			0			
#4			0			primeiro retorno
#3	1		2 + 0			
#4			2			segundo retorno
#3	2		4 + 2			
			. . .			
#4			(16+...+2)			penúltimo retorno
#3	9		18+(16+...+2)			
#4			(18+16+...+2)			último retorno
#3	10		20+(18+...+2)			
#4			(20+18+...+4+2)			
#5			(2+4+... +18+20)			parte principal

Programa em Java:

```
/*
 * Exemplo 7
 * Calcular o somatorio dos 10 primeiros pares, recursivamente.
 */
using System;

class Exemplo_7
{
// definir funcao com parametro
   public static int SOMAR ( int X )
   {
   // 1.1. definir dado local
      int S;               // somatorio
   // 1.2. verificar se X esta' fora do limite
      if ( X == 0 )
      {
      // 1.2.1 nao ha' mais o que somar
         S = 0;
      }
      else
      {
      // 1.2.2 somar se for termo valido
         S = 2 * X + SOMAR ( X-1 );
      } // fim se
      return ( S );
   } // fim da funcao SOMAR ( )
//
// parte principal
//
   public static void main ( String [ ] args )
   {
   // 2. calcular e mostrar o somatorio
      Console.WriteLine ( "\nS(N) = " + SOMAR ( 10 ) );
   // pausa para terminar
      Console.Write ( "\nPressionar ENTER para terminar." );
      Console.ReadLine ( );
   } // end Main ( )

} // fim Exemplo_7 class
```

Programa em Java:

```
/**
  * Exemplo 7
  * Calcular o somatorio dos 10 primeiros pares, recursivamente.
  */

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

class Exemplo_7
{
// definir funcao com parametro
   public static int SOMAR ( int X )
   {
   // 1.1. definir dado local
      int S;                              // somatorio
   // 1.2. verificar se X esta' fora do limite
      if ( X == 0 )
      {
      // 1.2.1 nao ha' mais o que somar
         S = 0;
      }
      else
      {
      // 1.2.2 somar se for termo valido
         S = 2 * X + SOMAR ( X-1 );
      } // fim se
      return ( S );
   } // fim da funcao SOMAR ( )
//
// parte principal
//
   public static void main ( String [ ] args )
   {
   // 2. calcular e mostrar o somatorio
      IO.println ( "\nS(N) = " + SOMAR ( 10 ) );
   // pausa para terminar
      IO.pause ( "\nPressionar ENTER para terminar." );
   } // end main ( )

} // fim Exemplo_7 class
```

Exemplo 8 :

Calcular a soma dos 10 primeiros números ímpares por uma função recursiva.

S(N) = 1 + 3 + ... + 17 + 19

Análise do problema:

O somatório pode ser definido recursivamente como:

$$S(X) = \begin{cases} 0, & \text{se } X \leq 0 \\ (2*X-1) + S(X-1), & \text{se } X \in [1:10] \end{cases}$$

- Análise de dados:

- Dados da função:

Dado	Tipo	Valor Inicial	Obs.
X	Inteiro		valor da vez
S	Real		somatório

- Avaliação da solução :

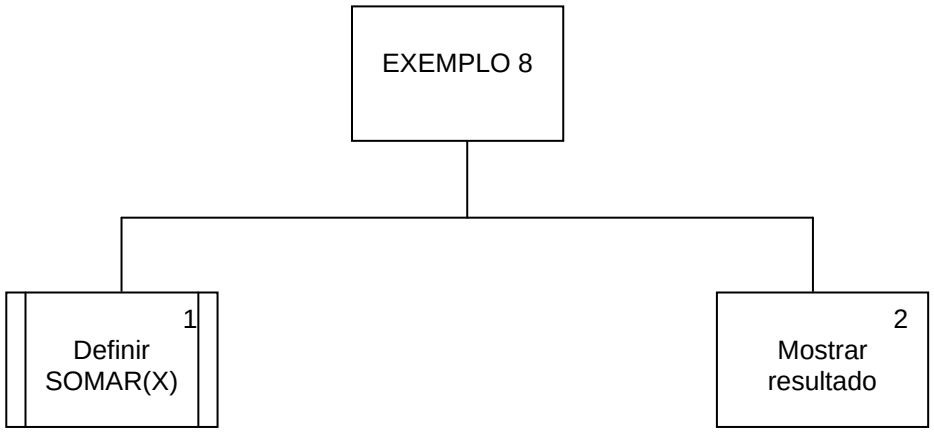
Calculando pela fórmula :

$$\sum_{i-1}^{n} 2i - 1 = 2\sum_{i-1}^{n} i - \sum_{i-1}^{n} 1 = 2\frac{n(n+1)}{2} - n = \quad n(n+1) - n = n^2 \Rightarrow \sum_{i-1}^{10} 2i - 1 = 10^2 = 100$$

Resultado :

SOMA = 100

Diagrama funcional:



Algoritmo, com função:

Esboço:

Primeira versão, só comentários.

Exemplo 8	v.1
Ação	Bloco
! definir função SOMAR (inteiro X)	1
! parte principal	
! calcular e mostrar o somatório	2

Segunda versão, com refinamento do segundo bloco.

Exemplo 8	v.2
Ação	Bloco
! definir função SOMAR (inteiro X)	1
! parte principal	
! calcular e mostrar o somatório tela ← ("S = ", SOMAR (10));	2

Terceira versão, refinar o primeiro bloco.

Exemplo 8			v.3
Ação			Bloco
! definir função			1
inteiro SOMAR (inteiro X)			
! definir dado local inteiro S;			1.1
! verificar se X está fora do limite			1.2
X==0?	V	S = 0;	1.2.1
	F	S= (2*X-1) + SOMAR(X-1);	1.2.2
retornar (S);			
! parte principal			
! calcular e mostrar o somatório tela ← ("S = ", SOMAR (10));			2

Programa em SCILAB:

```
// Exemplo 8.
// Calcular o somatorio dos 10 primeiros impares, recursivamente
//
// definir funcao com parametro
function retorno = SOMAR ( X )
// parametro:
// int X;                    // natural
//
// 1.1 definir dado local
   S = 0;                    // somatorio
// 1.2 verificar se X esta' fora do limite
   if ( X == 0 )
   // 1.2.1 nao ha' mais o que somar
      S = 0;                 // comecar a retornar
   else
   // 1.2.2 somar se o termo for valido
      S = (2*X-1) + SOMAR ( X-1 ); // proxima chamada
   end // fim se
// retornar o resultado
   retorno = S;
endfunction // SOMAR( )
//
// parte principal
//
// 2. calcular e mostrar o somatorio
   clc;   // limpar a area de trabalho
   printf ( "\nS(N) = %d ", SOMAR ( 10 ) );
// pausa para terminar
   printf ( "\nPressionar qualquer tecla para terminar.\n" );
   halt;
// fim do programa
```

Programa em C++:

```
// Exemplo 8.
// Calcular o somatorio dos 10 primeiros impares, recursivamente.
//
// bibliotecas necessarias
#include <iostream>
using namespace std;
//                                                  // LINHAS DE PROGRAMA
// definir funcao com parametro
int SOMAR ( int X )
{
 // 1.1 definir dado local
   int S;                          // somatorio
 // 1.2 verificar se X esta' fora do limite
   if ( X == 0 )                                    // # 1
   {
     // 1.2.1 nao ha mais o que somar
     S = 0;                        // comecar a retornar     // # 2
   }
   else
   {
     // 1.2.2 somar se for termo valido
      S = (2*X-1) + SOMAR ( X-1 );                  // # 3
   } // fim se
   return ( S );                                    // # 4
} // fim da funcao SOMAR ( )
//
// parte principal
//
int main ( void )
{
// 2. calcular e mostrar o somatorio
   cout << "\nS(N) = " << SOMAR ( 10 );             // #5
// pausa para terminar
   cout << "\nPressionar qualquer tecla para terminar.";
   getchar ( );
   return EXIT_SUCCESS;
} // fim do programa
```

Execução :

Parâmetros:	X		SOMAR	teclado	tela	Observação
Linha :			S			
#5	-					
#6			SOMAR(10)			primeira chamada
#1	10	X==0 ? F	0			
#3			19+SOMAR(9)			segunda chamada
#1	9	X==0 ? F	0			
#3			17+SOMAR(8)			terceira chamada
#1	8	X==0 ? F	0			
#3			15+SOMAR(7)			quarta chamada
			0			
			. . .			
#1	1	X==0 ? F				
#3			1+SOMAR(0)			última chamada
#1	0	X==0 ? F				
#2			0			
#4			0			primeiro retorno
#3	1		1 + 0			
#4			1			segundo retorno
#3	2		3 + 1			
			. . .			
#4			(15+...+1)			penúltimo retorno
#3	9		17+(15+...+1)			
#4			(17+15+...+1)			último retorno
#3	10		19+(17+...+1)			
#4			(19+17+...+3+1)			
#5			(1+3+... +17+19)			parte principal

Programa em C#:

```
/*
 * Exemplo 8
 * Calcular o somatorio dos 10 primeiros impares, recursivamente.
 */
using System;

class Exemplo_8
{
// definir funcao com parametro
   public static int SOMAR ( int X )
   {
   // 1.1. definir dado local
      int S;               // somatorio
   // 1.2. verificar se X esta' fora do limite
      if ( X == 0 )
      {
      // 1.2.1 nao ha' mais o que somar
         S = 0;
      }
      else
      {
      // 1.2.2 somar se for termo valido
         S = ( 2 * X - 1 ) + SOMAR ( X-1 );
      } // fim se
      return ( S );
   } // fim da funcao SOMAR ( )
//
// parte principal
//
   public static void Main ( )
   {
   // 2. calcular e mostrar o somatorio
      Console.WriteLine ( "\nS(N) = " + SOMAR ( 10 ) );
   // pausa para terminar
      Console.Write ( "\nPressionar ENTER para terminar." );
      Console.ReadLine ( );
   } // end Main ( )

} // fim Exemplo_8 class
```

Programa em Java:

```
/**
  * Exemplo 8
  * Calcular o somatorio dos 10 primeiros impares, recursivamente.
  */

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

class Exemplo_8
{
// definir funcao com parametro
   public static int SOMAR ( int X )
   {
   // 1.1. definir dado local
      int S;                              // somatorio
   // 1.2. verificar se X esta' fora do limite
      if ( X == 0 )
      {
      // 1.2.1 nao ha' mais o que somar
         S = 0;
      }
      else
      {
      // 1.2.2 somar se for termo valido
         S = ( 2 * X - 1 ) + SOMAR ( X-1 );
      } // fim se
      return ( S );
   } // fim da funcao SOMAR ( )
//
// parte principal
//
   public static void main ( String [ ] args )
   {
   // 2. calcular e mostrar o somatorio
      IO.println ( "\nS(N) = " + SOMAR ( 10 ) );
   // pausa para terminar
      IO.pause ( "\nPressionar ENTER para terminar." );
   } // end main ( )

} // fim Exemplo_8 class
```

Exemplo 9 :

Calcular o somatório abaixo por uma função recursiva para (N) termos.

$$S(N) = \frac{1}{1} + \frac{1}{2} + \frac{1}{3} - \cdots \frac{1}{n}$$

Análise do problema:

O somatório pode ser definido recursivamente como:

$$S(X) = \begin{cases} 0, & \text{se } X \leq 0 \\ 1/X + S(X-1), & \text{se } X > 0 \end{cases}$$

Exemplo:

N = 4

S(5) = 1/1 + 1/2 + 1/3 + 1/4

- Análise de dados:

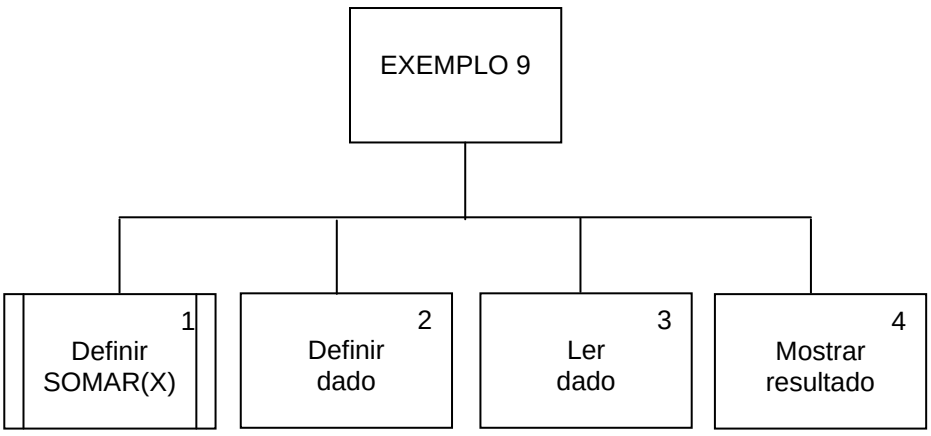
- Dados da função:

Dado	Tipo	Valor Inicial	Obs.
X	inteiro		valor da vez
S	real		somatório

- Dados da parte principal:

Dado	Tipo	Valor Inicial	Obs.
N	inteiro		número de termos

Diagrama funcional:



Algoritmo, com função:

Esboço:

Primeira versão, só comentários.

Exemplo 9	v.1
Ação	Bloco
! definir função SOMAR (inteiro X)	1
! parte principal	
! definir dados locais	2
! ler dados	3
! calcular e mostrar o somatório	4

Segunda versão, com refinamento do segundo, terceiro e quarto blocos.

Exemplo 9	v.2
Ação	Bloco
! definir função SOMAR (inteiro X)	1
! parte principal	
! definir dados locais inteiro N; ! número de termos	2
! ler dado tela ← "Quantos termos ? "; N ← teclado;	3
! calcular e mostrar o somatório tela ← ("S = ", SOMAR (N));	4

Terceira versão, refinar o primeiro bloco.

Exemplo 9			v.3
Ação			Bloco
! definir função			1
real SOMAR (inteiro X)			
! definir dado local real S;			1.1
! verificar se X é nulo			1.2
X==0?	V	S = 0;	1.2.1
	F	S= 1.0/X+SOMAR(X-1);	1.2.2
retornar (S);			
! parte principal			
! definir dado local inteiro N; ! número de termos			2
! ler dado tela ← "Quantos termos ? "; N ← teclado;			3
! calcular e mostrar o somatório tela ← ("S = ", SOMAR (N));			4

Programa em SCILAB:

```
// Exemplo 9.
// Calcular o somatorio dos inversos dos N primeiros naturais, recursivamente
//
// definir funcao com parametro
function retorno = SOMAR ( X )
// parametro:
// int X;                       // natural
//
// 1.1 definir dado local
   S = 0.0;                     // somatorio
// 1.2 verificar se X e' nulo
   if ( X == 0 )
   // 1.2.1 nao ha' mais o que somar
      S = 0.0;                  // comecar a retornar
   else
   // 1.2.2 somar se o termo for valido
      S = 1.0 / X + SOMAR ( X - 1 );        // proxima chamada
   end // fim se
// retornar o resultado
   retorno = S;
endfunction // SOMAR( )
//
// parte principal
//
// 2. definir dado local
   N = 0;                       // numero de termos
// 3. ler dado
   clc;         // limpar a area de trabalho
   N = input ( "\nQuantos termos ? " );
// 4. calcular e mostrar o somatorio
   printf ( "\nS(%d) = %f ", N, SOMAR ( N ) );
// pausa para terminar
   printf ( "\nPressionar qualquer tecla para terminar.\n" );
   halt;
// fim do programa
```

Programa em C++:

```
// Exemplo 9.
// Calcular o somatorio dos inversos dos N primeiros naturais, recursivamente.
//
// bibliotecas necessárias
#include <iostream>
using namespace std;
//                                                    // LINHAS DE PROGRAMA
// definir funcao com parametros
float SOMAR ( int X )
{
 // 1.1 definir dado local
   float S;                        // somatório
 // 1.2 verificar se X e' nulo
   if ( X == 0 )                                     // # 1
   {
     // 1.2.1 não há mais o que somar
     S = 0.0;                      // comecar a retornar   // # 2
   }
   else
   {
     // 1.2.2 se for termo impar
      S = 1.0 / X + SOMAR ( X-1 );                   // # 3
   } // fim se
   return ( S );                                     // # 4
} // fim da funcao SOMAR ( )
//
// parte principal
//
int main ( void )
{
// 2. definir dado local
   int N,                          // numero de termos
// 3. ler dado
   cout << "\nQual o valor de N ? ";   cin >> N;      // #5
// 4. calcular e mostrar o somatorio
   cout << "\nS(N) = " << SOMAR ( N );                // #6
// pausa para terminar
   cout << "\nPressionar qualquer tecla para terminar.";
   getchar ( );
   return EXIT_SUCCESS;
} // fim do programa
```

Execução :

Dados : N = 4

Parâmetros: Linha :	X		SOMARS	teclado	tela	Observação
#5	-	-			4	
#6			SOMAR(4)			primeira chamada
#1		X = 0 ? F				
#3	4		1/4+SOMAR(3)			segunda chamada
#1		X = 0 ? F				
#3	3		1/3+SOMAR(2)			terceira chamada
#1		X = 0 ? F				
#3	2		1/2+SOMAR(1)			quarta chamada
#1		X = 0 ? F				
#3	1		1/1+SOMAR(0)			quinta chamada
#1		X = 0 ? F				
#2	0		0			
#4	0		0			primeiro retorno
#3	1		1/1+0			
#4	1		1/1			segundo retorno
#3	2		1/2+1/1			
#4	2		(1/2+1/1)			terceiro retorno
#3	3		1/3+(1/2-1/1)			
#4	3		(1/3+1/2-1/1)			quarto retorno
#3	4		1/4+(1/31/2+1/1)			
#4	4		(1/4+1/3+1/2+1/1)			quinto retorno
#6			(1/1+1/2+1/3+1/4)			parte principal

Programa em C#:

```
/*
 * Exemplo  9
 * Calcular o somatorio dos inversos dos N primeiros numeros naturais, recursivamente.
 */
using System;

class Exemplo_9
{
// definir funcao com parametros
   public static double SOMAR ( int X )
   {
   // 1.1. definir dado local
      double S;                    // somatorio
   // 1.2. verificar se X e' nulo
      if ( X == 0 )
      {
      // 1.2.1 nao ha' mais o que somar
         S = 0.0;
      }
      else
      {
      // 1.2.2 somar se for termo valido
         S = 1.0 / X + SOMAR ( X-1 );
      } // fim se
      return ( S );
   } // fim da funcao SOMAR ( )
//
// parte principal
//
   public static void Main ( )
   {
   // 2. definir dado
      int N;                       // numero de termos
   // 3. ler um valor inteiro
      Console.Write ( "\nQual o valor de N ?" );
      N = int.Parse ( Console.ReadLine ( ) );
   // 4. calcular e mostrar o somatorio
      Console.WriteLine ("\nS(N) = " + SOMAR ( N ) );
   // pausa para terminar
      Console.Write ( "\nPressionar ENTER para terminar." );
      Console.ReadLine ( );
   } // end Main ( )

} // fim Exemplo_9 class
```

Programa em Java:

```
/**
  * Exemplo  9
  * Calcular o somatorio dos inversos dos N primeiros numeros naturais, recursivamente.
  */

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

class Exemplo_9
{
// definir funcao com parametros
   public static double SOMAR ( int X )
   {
   // 1.1. definir dado local
      double S;                    // somatorio
   // 1.2. verificar se X e' nulo
      if ( X == 0 )
      {
      // 1.2.1 nao ha' mais o que somar
         S = 0.0;
      }
      else
      {
      // 1.2.2 somar se for termo valido
         S = 1.0 / X + SOMAR ( X-1 );
      } // fim se
      return ( S );
   } // fim da funcao SOMAR ( )
//
// parte principal
//
   public static void main ( String [ ] args )
   {
   // 2. definir dado
      int N;               // numero de termos
   // 3. ler dado
      N = IO.readint ( "\nQual o valor de N ? " );
   // 4. calcular e mostrar o somatorio
      IO.println ( "\nS(N) = " + SOMAR ( N ) );
   // pausa para terminar
      IO.pause ( "\nPressionar ENTER para terminar." );
   } // end main ( )

} // fim Exemplo_9 class
```

Exemplo 10 :

Calcular o somatório abaixo por uma função recursiva para (N) termos.

$$S(N) = \frac{1}{n} - \frac{1}{n-1} + \frac{1}{n-2} - \cdots - 1$$

Análise do problema:

O somatório pode ser definido recursivamente como:

S (X) =
- 1/X + S (X-1) , se X > 0 e termo par
- 0 , se X = 0
- -1/X + S (X-1) , se X > Y e termo ímpar

Exemplo:

N = 4

S(5) = 1/4 - 1/3 + 1/2 - 1

- Análise de dados:

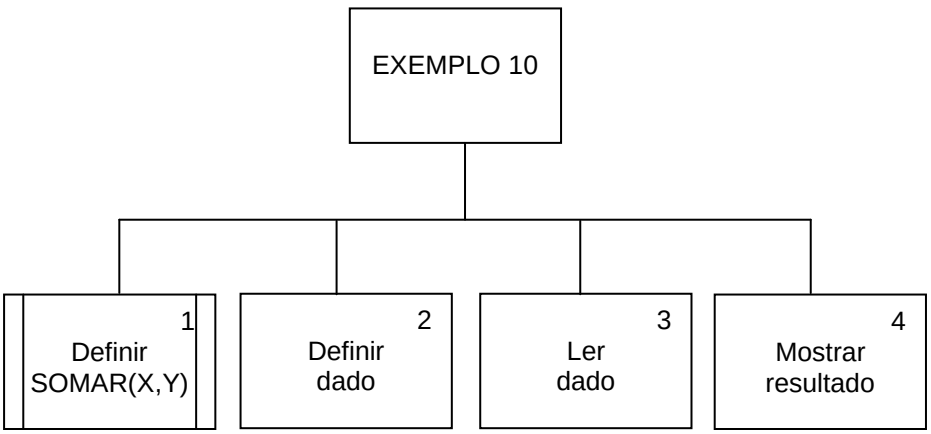
- Dados da função:

Dado	Tipo	Valor Inicial	Obs.
X	inteiro		valor da vez
Y	inteiro		ordem do termo
S	real		somatório

- Dados da parte principal:

Dado	Tipo	Valor Inicial	Obs.
N	inteiro		número de termos

Diagrama funcional:



Algoritmo, com função:

Esboço:

Primeira versão, só comentários.

Exemplo 10	v.1
Ação	Bloco
! definir função SOMAR (inteiro X, inteiro Y)	1
! parte principal	
! definir dados locais	2
! ler dados	3
! calcular e mostrar o somatório	4

Segunda versão, com refinamento do segundo, terceiro e quarto blocos.

Exemplo 10	v.2
Ação	Bloco
! definir função SOMAR (inteiro X, inteiro Y)	1
! parte principal	
! definir dados locais inteiro N; ! número de termos	2
! ler dado tela ← "Quantos termos ? "; N ← teclado;	3
! calcular e mostrar o somatório tela ← ("S = ", SOMAR (N, 1));	4

Terceira versão, refinar o primeiro bloco.

Exemplo 10						v.3
Ação						Bloco
! definir função						1
real SOMAR (inteiro X, inteiro Y)						
! definir dado local real S;						1.1
! verificar se X é nulo						1.2
X==0?	V	S = 0;				1.2.1
	F	Y//2==0?	V	S= - 1.0/X +SOMAR(X-1,Y+1);		1.2.2
			F	S= 1.0/X +SOMAR(X-1,Y+1);		1.2.3
retornar (S);						
! parte principal						
! definir dados locais inteiro N; ! número de termos						2
! ler dado tela ← "Quantos termos ? "; N ← teclado;						3
! calcular e mostrar o somatório tela ← ("S = ", SOMAR (N, 1));						4

Programa em C++:

```
// Exemplo 10a.
// Calcular um somatorio com alternancia de sinal, recursivamente.
//
// bibliotecas necessarias
#include <iostream>
using namespace std;
//                                                  // LINHAS DE PROGRAMA
// definir funcao com parametro
float SOMAR ( int X, int Y )
{
 // 1.1 definir dado local
   float S;                        // somatorio
 // 1.2 verificar se X e' nulo
   if ( X == 0 )                                    // # 1
   {
     // 1.2.1 nao ha' mais o que somar
     S = 0.0;                      // começar a retornar     // # 2
   }
   else
   {
     if ( Y // 2 == 0 )                             // # 3
     {
      // 1.2.2 se for termo par
      S = - 1.0/X + SOMAR ( X-1, Y+1 );             // # 4
     }
     else
     {
      // 1.2.3 se for termo ímpar
      S =   1.0/X + SOMAR ( X-1, Y+1 );             // # 5
     } // fim se
   } // fim se
   return ( S );                                    // # 6
} // fim da funcao SOMAR ( )
//
// parte principal
//
int main ( void )
{
// 2. definir dado
   int N,                          // numero de termos
// 3. ler dado
   cout << "\nQual o valor de N ? ";   cin >> N;          // #7
// 4. calcular e mostrar o somatorio
   cout << "\nS(N) = " << SOMAR ( N, 1 );                // #8
// pausa para terminar
   cout << "\nPressionar qualquer tecla para terminar.";
   getchar ( );
   return EXIT_SUCCESS;
} // fim do programa
```

Outra solução:

- Análise de dados:

- Dados da função:

Dado	Tipo	Valor Inicial	Obs.
X	inteiro		valor da vez
Sinal	inteiro		sinal do termo
S	real		somatório

Esboço:

Primeira versão, só comentários.

Exemplo 10	v.1
Ação	Bloco
! definir função SOMAR (inteiro X, inteiro Sinal)	1
! parte principal	
! definir dados locais	2
! ler dados	3
! calcular e mostrar o somatório	4

Segunda versão, manter o resto e refinar o primeiro bloco.

Exemplo 10			v.3
Ação			Bloco
! definir função			1
real SOMAR (inteiro X, inteiro Sinal)			
! definir dado local real S;			1.1
! verificar se X é nulo			1.2
X==0?	V	! não há mais o que somar S=0;	1.2.1
	F	! somar um termo com o resto S=Sinal * 1.0/X + SOMAR(X-1,-Sinal);	1.2.2
retornar (S);			
! parte principal			
! definir dados locais inteiro N; ! número de termos			2
! ler dado tela ← "Quantos termos ? "; N ← teclado;			3
! calcular e mostrar o somatório tela ← ("S = ", SOMAR (N, 1));			4

Programa em C++:

```
// Exemplo 10b.
// Calcular um somatorio com alternancia de sinal, recursivamente.
//
// bibliotecas necessarias
#include <iostream>
using namespace std;
//                                                  // LINHAS DE PROGRAMA
// definir funcao com parametros
float SOMAR ( int X, int Sinal )
{
 // 1.1 definir dado local
   float S;                       // somatorio
 // 1.2 verificar se X e' nulo
   if ( X == 0 )                                    // # 1
   {
     // 1.2.1 nao ha' mais o que somar
     S = 0;                       // comecar a retornar    // # 2
   }
   else
   {
     // 1.2.2 somar um termo com o resto
      S = Sinal * 1.0/X + SOMAR ( X-1, -Sinal );    // # 3
   } // fim se
   return ( S );                                    // # 4
} // fim da funcao SOMAR ( )
//
// parte principal
//
int main ( void )
{
// 2. definir dado
   int N,                         // numero de termos
// 3. ler dado
   cout << "\nQual o valor de N ? ";   cin >> N;     // #5
// 4. calcular e mostrar o somatorio
   cout << "\nS(N) = " << SOMAR ( N, 1 );           // #6
// pausa para terminar
   cout << "\nPressionar qualquer tecla para terminar.";
   getchar ( );
   return EXIT_SUCCESS;
} // fim do programa
```

Uma terceira solução:

- Análise de dados:

- Dados da função:

Dado	Tipo	Valor Inicial	Obs.
X	inteiro		valor da vez
Expoente	inteiro		sinal do termo
S	real		somatório

Esboço:

Primeira versão, só comentários.

Exemplo 10	v.1
Ação	Bloco
! definir função SOMAR (inteiro X, inteiro Sinal)	1
! parte principal	
! definir dados locais	2
! ler dados	3
! calcular e mostrar o somatório	4

Segunda versão, manter o resto e refinar o primeiro bloco.

Exemplo 10			v.2
Ação			Bloco
! definir função			1
real SOMAR (inteiro X, inteiro Expoente)			
! definir dado local real S;			1.1
! verificar se X é nulo			1.2
X==0?	V	! não há mais o que somar S = 0;	1.2.1
	F	! somar um termo com o resto S = (-1 ^ Expoente) / X + SOMAR(X-1,-Expoente+1);	1.2.2
retornar (S);			
! parte principal			
! definir dados locais inteiro N; ! número de termos			2
! ler dado tela ← "Quantos termos ? "; N ← teclado;			3
! calcular e mostrar o somatório tela ← ("S = ", SOMAR (N, 2));			4

Programa em SCILAB:

```
// Exemplo 10c.
// Calcular um somatorio com alternancia de sinal, recursivamente.
//
// definir funcao com parametros
function retorno = SOMAR ( X, Expoente )
// parametro:
// int X;                       // natural
// int Expoente;
//
// 1.1 definir dado local
   S = 0;                       // somatorio
// 1.2 verificar se X e' nulo
   if ( X == 0 )
   // 1.2.1 nao ha' mais o que somar
      S = 0;                    // comecar a retornar
   else
   // 1.2.2 somar sinal e o termo com o resto
      S = ((-1) ^ Expoente)/X + SOMAR ( X-1, Expoente+1 );
   end // fim se
// retornar o resultado
   retorno = S;
endfunction  // SOMAR( )
//
//
// parte principal
//
// 2. definir dadol
   N = 0;                       // numero de termos
// 3. ler dado
   clc;         // limpar a area de trabalho
   N = input ( "\nQual o valor de N ? " );
// 4. calcular e mostrar o somatorio
   printf ( "\nS(N) = %f ", SOMAR ( N, 2 ) );
// pausa para terminar
   printf ( "\nPressionar qualquer tecla para terminar.\n" );
   halt;
// fim do programa
```

Programa em C++:

```
// Exemplo 10c.
// Calcular um somatorio com alternancia de sinal, recursivamente.
//
// bibliotecas necessarias
#include <iostream>
using namespace std;
//                                                          // LINHAS DE PROGRAMA
// definir funcao com parametros
float SOMAR ( int X, int Expoente )
{
 // 1.1 definir dado local
   float S;                        // somatorio
 // 1.2 verificar se X e' nulo
   if ( X == 0 )                                          // # 1
   {
     // 1.2.1 nao ha' mais o que somar
     S = 0;                        // comecar a retornar  // # 2
   }
   else
   {
     // 1.2.2 somar sinal e termo com o resto
      S = pow (-1, Expoente) / X+SOMAR ( X-1,Expoente+1 );// # 3
   } // fim se
   return ( S );                                          // # 4
} // fim da funcao SOMAR ( )
//
// parte principal
//
int main ( void )
{
// 2. definir dado
   int N,                          // número de termos
// 3. ler dado
   cout << "\nQual o valor de N ? ";   cin >> N;          // #5
// 4. calcular e mostrar o somatorio
   cout << "\nS(N) = " << SOMAR ( N, 2 );                // #6
// pausa para terminar
   cout << "\nPressionar qualquer tecla para terminar.";
   getchar ( );
   return EXIT_SUCCESS;
} // fim do programa
```

Execução :

Dados : N = 4

Parâmetros: Linha :	X	Expoente	SOMARS	teclado	tela	Observação
#5	-	-			4	
#6			SOMAR(4, 2)			primeira chamada
#1		X = 0 ? F				
#3	4	1	1/4+SOMAR(3,3)			segunda chamada
#1		X = 0 ? F				
#3	3	-1	-1/3+SOMAR(2,4)			terceira chamada
#1		X = 0 ? F				
#3	2	1	1/2+SOMAR(1,5)			quarta chamada
#1		X = 0 ? F				
#3	1	-1	-1/1+SOMAR(0,6)			quinta chamada
#1		X = 0 ? F				
#2	0	6	0			
#4	0	6	0			primeiro retorno
#3	1	5	-1/1+0			
#4	1	5	-1			segundo retorno
#3	2	4	1/2-1			
#4	2	4	(1/2-1)			terceiro retorno
#3	3	3	-1/3+(1/2-1)			
#4	3	3	(-1/3+1/2-1)			quarto retorno
#3	4	2	1/4+(-1/31/2-1)			
#4	4	2	(1/4-1/3+1/2-1)			quinto retorno
#6			(1/4-1/3+1/2-1)			parte principal

Programa em C#:

```
/*
 * Exemplo 10a
 * Calcular um somatorio com alternancia de sinal, recursivamente.
 */
using System;

class Exemplo_10a
{
// definir funcao com parametro
   public static double SOMAR ( int X, int Y )
   {
   // 1.1. definir dado local
      double S;                    // somatorio
   // 1.2. verificar se X esta' fora do limite
      if ( X == 0 )
      {
      // 1.2.1 nao ha' mais o que somar
         S = 0.0;
      }
      else
      {
               if ( Y // 2 == 0 )
               {
        // 1.2.2 se for termo par
           S = -1.0 / X + SOMAR ( X-1, Y+1 );
         }
         else
               {
        // 1.2.3 se for termo impar
           S =  1.0 / X + SOMAR ( X-1, Y+1 );
         } // fim se
      } // fim se
      return ( S );
   } // fim da funcao SOMAR ( )
//
// parte principal
//
   public static void Main ( )
   {
   // 2. definir dado
      int N;               // numero de termos
   // 3. ler dado
      Console.Write ( "\nQual o valor de N ?" );
      N = int.Parse ( Console.ReadLine ( ) );
   // 4. calcular e mostrar o somatorio
      Console.WriteLine ("\nS(N) = " + SOMAR ( N, 1 ) );
   // pausa para terminar
      Console.Write ( "\nPressionar ENTER para terminar." );
      Console.ReadLine ( );
   } // end Main ( )

} // fim Exemplo_10a class
```

Outra versão do programa em C#:

```
/*
 * Exemplo 10b
 * Calcular um somatorio com alternancia de sinal, recursivamente.
 */
using System;

class Exemplo_10b
{
// definir funcao com parametro
   public static double SOMAR ( int X, int Sinal )
   {
   // 1.1. definir dado local
      double S;                    // somatorio
   // 1.2. verificar se X esta' fora do limite
      if ( X == 0 )
      {
      // 1.2.1 nao ha' mais o que somar
         S = 0.0;
      }
      else
      {
      // 1.2.2 somar um termo com o resto
         S = Sinal * 1.0 / X + SOMAR ( X-1, -Sinal );
      } // fim se
      return ( S );
   } // fim da funcao SOMAR ( )
//
// parte principal
//
   public static void Main ( )
   {
   // 2. definir dado
      int N;               // numero de termos
   // 3. ler dado
      Console.Write ( "\nQual o valor de N ?" );
      N = int.Parse ( Console.ReadLine ( ) );
   // 4. calcular e mostrar o somatorio
      Console.WriteLine ("\nS(N) = " + SOMAR ( N, 1 ) );
   // pausa para terminar
      Console.Write ( "\nPressionar ENTER para terminar." );
      Console.ReadLine ( );
   } // end Main ( )

} // fim Exemplo_10b class
```

Uma terceira versão do programa em C#:

```
/*
 * Exemplo 10c
 * Calcular um somatorio com alternancia de sinal, recursivamente.
 */
using System;

class Exemplo_10c
{
// definir funcao com parametro
   public static double SOMAR ( int X, int Expoente )
   {
   // 1.1. definir dado local
      double S;                    // somatorio
   // 1.2. verificar se X esta' fora do limite
      if ( X == 0 )
      {
      // 1.2.1 nao ha' mais o que somar
         S = 0.0;
      }
      else
      {
      // 1.2.2 somar sinal e termo com o resto
         S = Math.pow (-1, Expoente) / X + SOMAR ( X-1, Expoente+1 );
      } // fim se
      return ( S );
   } // fim da funcao SOMAR ( )
//
// parte principal
//
   public static void Main ( )
   {
   // 2. definir dado
      int N;               // numero de termos
   // 3. ler dado
      Console.Write ( "\nQual o valor de N ?" );
      N = int.Parse ( Console.ReadLine ( ) );
   // 4. calcular e mostrar o somatorio
      Console.WriteLine ("\nS(N) = " + SOMAR ( N, 2 ) );
   // pausa para terminar
      Console.Write ( "\nPressionar ENTER para terminar." );
      Console.ReadLine ( );
   } // end Main ( )

} // fim Exemplo_10c class
```

Programa em Java:

```
/**
  * Exemplo 10a
  * Calcular um somatorio com alternancia de sinal, recursivamente.
  */

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

class Exemplo_10a
{
// definir funcao com parametro
   public static double SOMAR ( int X, int Y )
   {
   // 1.1. definir dado local
      double S;                    // somatorio
   // 1.2. verificar se X esta' fora do limite
      if ( X == 0 )
      {
      // 1.2.1 nao ha' mais o que somar
         S = 0.0;
      }
      else
      {
               if ( Y // 2 == 0 )
               {
        // 1.2.2 se for termo par
           S = -1.0 / X + SOMAR ( X-1, Y+1 );
         }
         else
               {
        // 1.2.3 se for termo impar
           S =  1.0 / X + SOMAR ( X-1, Y+1 );
         } // fim se
      } // fim se
      return ( S );
   } // fim da funcao SOMAR ( )
//
// parte principal
//
   public static void main ( String [ ] args )
   {
   // 2. definir dado
      int N;               // numero de termos
   // 3. ler dado
      N = IO.readint ( "\nQual o valor de N ? " );
   // 4. calcular e mostrar o somatorio
      IO.println ( "\nS(N) = " + SOMAR ( N, 1 ) );
   // pausa para terminar
      IO.pause ( "\nPressionar ENTER para terminar." );
   } // end main ( )

} // fim Exemplo_10a class
```

Outra versão do programa em Java:

```
/**
  * Exemplo 10b
  * Calcular um somatorio com alternancia de sinal, recursivamente.
  */

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

class Exemplo_10b
{
// definir funcao com parametro
   public static double SOMAR ( int X, int Sinal )
   {
   // 1.1. definir dado local
      double S;                    // somatorio
   // 1.2. verificar se X esta' fora do limite
      if ( X == 0 )
      {
      // 1.2.1 nao ha' mais o que somar
         S = 0.0;
      }
      else
      {
      // 1.2.2 somar um termo com o resto
         S = Sinal * 1.0 / X + SOMAR ( X-1, -Sinal );
      } // fim se
      return ( S );
   } // fim da funcao SOMAR ( )
//
// parte principal
//
   public static void main ( String [ ] args )
   {
   // 2. definir dado
      int N;               // numero de termos
   // 3. ler dado
      N = IO.readint ( "\nQual o valor de N ? " );
   // 4. calcular e mostrar o somatorio
      IO.println ( "\nS(N) = " + SOMAR ( N, 1 ) );
   // pausa para terminar
      IO.pause ( "\nPressionar ENTER para terminar." );
   } // end main ( )

} // fim Exemplo_10b class
```

Uma terceira versão do programa em Java:

```
/**
  * Exemplo 10c
  * Calcular um somatorio com alternancia de sinal, recursivamente.
  */

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

class Exemplo_10c
{
// definir funcao com parametro
   public static double SOMAR ( int X, int Expoente )
   {
   // 1.1. definir dado local
      double S;                       // somatorio
   // 1.2. verificar se X esta' fora do limite
      if ( X == 0 )
      {
      // 1.2.1 nao ha' mais o que somar
         S = 0.0;
      }
      else
      {
      // 1.2.2 somar sinal e termo com o resto
         S = Math.pow (-1, Expoente) / X + SOMAR ( X-1, Expoente+1 );
      } // fim se
      return ( S );
   } // fim da funcao SOMAR ( )
//
// parte principal
//
   public static void main ( String [ ] args )
   {
   // 2. definir dado
      int N;                // numero de termos
   // 3. ler dado
      N = IO.readint ( "\nQual o valor de N ? " );
   // 4. calcular e mostrar o somatorio
      IO.println ( "\nS(N) = " + SOMAR ( N, 2 ) );
   // pausa para terminar
      IO.pause ( "\nPressionar ENTER para terminar." );
   } // end main ( )

} // fim Exemplo_10c class
```

Exercícios (mostrar a execução)

1. Escrever um algoritmo para :
   - ler do teclado um número inteiro (X);
   - calcular e mostrar o fatorial recursivo definido por :

$$\text{FATORIAL}(X) = \begin{cases} 1 & \text{se } X = 0 \\ X * \text{FATORIAL}(X-1) & \text{se } X > 0 \end{cases}$$

2. Escrever um algoritmo para :
   - ler do teclado dois números inteiros (X e Y);
   - calcular e mostrar o máximo divisor comum recursivo :

$$\text{MDC}(X,Y) = \begin{cases} \text{MDC}(Y,X), & \text{se } X < Y \\ X, & \text{se } X = Y \\ \text{MDC}(X-Y,Y), & \text{se } X > Y \end{cases}$$

3. Escrever um algoritmo para :
   - ler do teclado um número inteiro (N);
   - calcular e mostrar o termo (N) da série de Fibonacci :

$$\text{FIB}(N) = \begin{cases} 1 & \text{para } N = 1 \text{ e } 2 \\ \text{FIB}(N-1) + \text{FIB}(N-2) & \text{para } N > 2 \end{cases}$$

4. Utilizar a função acima em um algoritmo para :
   a.) gerar os 10 primeiros termos da série;
   b.) ler um inteiro (N) e gerar todos os termos menores que N;
   c.) ler dois números inteiros M e N e calcular a diferença entre FIB(M) e FIB(N) .

5. Escrever uma função recursiva para mostrar os divisores de um número.

6. Reescrever a função anterior para contar o número de divisores de um número.

7. Escrever um procedimento recursivo para calcular a soma dos divisores de um número.

8. Escrever um procedimento recursivo para mostrar a seguinte seqüência :

1024 - 512 - 256 - ... - 4 - 2 - 1

9. Escrever uma função recursiva para calcular e mostrar o valor de "S", com 20 termos :

$$S = 1 + 1/2 + 2/3 + 3/5 + ...$$

10. Escrever um procedimento recursivo capaz de verificar se um número é palíndromo.