

Tema: Introdução à programação II

Atividade: Funções e procedimentos iterativos em C

01.) Editar e salvar um esboço de programa em C, cujo nome será Exemplo0501.c, para ler e mostrar certa quantidade de valores:

```
/*
    Exemplo0501 - v0.0. - __ / __ / ____
    Author: _____
*/
// dependencias
#include "IO.h"          // para definicoes proprias

/**
    Method00 - nao faz nada.
*/
void method00 ( )
{
    // nao faz nada
} // fim method00 ( )

/**
    Method01a - Mostrar certa quantidade de valores.
    @param x - quantidade de valores a serem mostrados
*/
void method01a ( int x )
{
    // definir dado local
    int y = 1;           // controle

    // repetir enquanto controle menor que a quantidade desejada
    while ( y <= x )
    {
        // mostrar valor
        IO_printf ( "%s%d\n", "Valor = ", y );
        // passar ao proximo
        y = y + 1;
    } // fim se
} // fim method01a( )
```

```

/**
    Method01 - Mostrar certa quantidade de valores.
    OBS.: Preparacao e disparo de outro metodo.
*/
void method01 ( )
{
    // identificar
    IO_id ( "EXEMPLO0501 - Method01 - v0.0" );

    // executar o metodo auxiliar
    method01a ( 5 );

    // encerrar
    IO_pause ( "Apertar ENTER para continuar" );
} // fim method01 ( )

/**
    Method02.
*/
void method02 ( )
{
    // identificar
    IO_id ( "EXEMPLO0501 - Method02 - v0.0" );

    // encerrar
    IO_pause ( "Apertar ENTER para continuar" );
} // fim method02 ( )

/**
    Method03.
*/
void method03 ( )
{
    // identificar
    IO_id ( "EXEMPLO0501 - Method03 - v0.0" );

    // encerrar
    IO_pause ( "Apertar ENTER para continuar" );
} // fim method03 ( )

/**
    Method04.
*/
void method04 ( )
{
    // identificar
    IO_id ( "EXEMPLO0501 - Method04 - v0.0" );

    // encerrar
    IO_pause ( "Apertar ENTER para continuar" );
} // fim method04 ( )

```

```

/**
    Method05.
*/
void method05 ( )
{
    // identificar
    IO_id ( "EXEMPLO0501 - Method05 - v0.0" );

    // encerrar
    IO_pause ( "Apertar ENTER para continuar" );
} // fim method05 ( )

/**
    Method06.
*/
void method06 ( )
{
    // identificar
    IO_id ( "EXEMPLO0501 - Method06 - v0.0" );

    // encerrar
    IO_pause ( "Apertar ENTER para continuar" );
} // fim method06 ( )

/**
    Method07.
*/
void method07 ( )
{
    // identificar
    IO_id ( "EXEMPLO0501 - Method07 - v0.0" );

    // encerrar
    IO_pause ( "Apertar ENTER para continuar" );
} // fim method07 ( )

/**
    Method08.
*/
void method08 ( )
{
    // identificar
    IO_id ( "EXEMPLO0501 - Method08 - v0.0" );

    // encerrar
    IO_pause ( "Apertar ENTER para continuar" );
} // fim method08 ( )

/**
    Method09.
*/
void method09 ( )
{
    // identificar
    IO_id ( "EXEMPLO0501 - Method09 - v0.0" );

    // encerrar
    IO_pause ( "Apertar ENTER para continuar" );
} // fim method09 ( )

```

```

/**
    Method10.
*/
void method10 ( )
{
    // identificar
    IO_id ( "EXEMPLO0501 - Method10 - v0.0" );

    // encerrar
    IO_pause ( "Apertar ENTER para continuar" );
} // fim method10 ( )

/*
    Funcao principal.
    @return codigo de encerramento
*/
int main ( )
{
    // definir dado
    int x = 0;          // definir variavel com valor inicial

    // repetir até desejar parar
    do
    {
        // identificar
        IO_id ( "EXEMPLO0501 - Programa - v0.0" );

        // ler do teclado
        IO_println ( "Opcoes" );
        IO_println ( " 0 - parar" );
        IO_println ( " 1 - mostrar certa quantidade de valores" );
        IO_println ( " 2 - " );
        IO_println ( " 3 - " );
        IO_println ( " 4 - " );
        IO_println ( " 5 - " );
        IO_println ( " 6 - " );
        IO_println ( " 7 - " );
        IO_println ( " 8 - " );
        IO_println ( " 9 - " );
        IO_println ( "10 - " );
        IO_println ( "" );

        x = IO_readint ( "Entrar com uma opcao: " );

    // testar valor
    switch ( x )
    {
        case 0:
            method00 ( );
            break;
        case 1:
            method01 ( );
            break;
        case 2:
            method02 ( );
            break;
        case 3:
            method03 ( );
            break;
    }
}

```

```

    case 4:
        method04 ( );
        break;
    case 5:
        method05 ( );
        break;
    case 6:
        method06 ( );
        break;
    case 7:
        method07 ( );
        break;
    case 8:
        method08 ( );
        break;
    case 9:
        method09 ( );
        break;
    case 10:
        method10 ( );
        break;
    default:
        IO_printf ( "ERRO: Valor invalido." );
} // fim escolher
}
while ( x != 0 );

// encerrar
IO_pause ( "Apertar ENTER para terminar" );
return ( 0 );
} // fim main( )

/*
----- documentacao complementar

----- notas / observacoes / comentarios

----- previsao de testes

a.) 5 -> { 1, 2, 3, 4, 5 }

----- historico

Versao      Data      Modificacao
0.1         __/___     esboco

----- testes

Versao      Teste
0.1         01. ( OK )   identificacao de programa

*/

```

02.) Compilar o programa.

Se houver erros, resolvê-los e compilar novamente, até que todos tenham sido resolvidos.

Se não houver erros, seguir para o próximo passo.

Em caso de dúvidas, consultar a apostila, recorrer aos monitores ou apresentá-las ao professor.

03.) Executar o programa. Observar as saídas. Registrar os dados e os resultados.

Em caso de erro (ou dúvida), usar comentários para registrar a ocorrência e, posteriormente, tentar resolvê-lo (ou esclarecer a dúvida).

04.) Copiar a versão atual do programa para outra nova – Exemplo0502.c.

05.) Editar mudanças no nome do programa e versão.

Acrescentar métodos para mostrar valores inteiros em ordem crescente.

Na parte principal, editar a chamada do método de preparação e disparo de outro.

Prever novos testes.

```
/**
 * Method02a - Mostrar certa quantidade de valores pares.
 * @param x - quantidade de valores a serem mostrados
 */
void method02a ( int x )
{
    // definir dado local
    int y = 1;           // controle
    int z = 2;

    // repetir enquanto controle menor que a quantidade desejada
    while ( y <= x )
    {
        // mostrar valor
        IO_printf ( "%d: %d\n", y, z );
        // passar ao proximo par
        z = z + 2;
        // passar ao proximo valor controlado
        y = y + 1;
    } // fim se
} // fim method02a( )

/**
 * Method02.
 */
void method02 ( )
{
    // identificar
    IO_id ( "EXEMPLO0502 - Method02 - v0.0" );

    // executar o metodo auxiliar
    method02a ( 5 );

    // encerrar
    IO_pause ( "Apertar ENTER para continuar" );
} // fim method02 ( )
```

- 06.) Compilar o programa novamente.
Se houver erros, resolvê-los e compilar novamente, até que todos tenham sido resolvidos.
Se não houver erros, seguir para o próximo passo.
- 07.) Executar o programa. Observar as saídas. Registrar os dados e os resultados.
- 08.) Copiar a versão atual do programa para outra nova – Exemplo0503.c.
- 09.) Editar mudanças no nome do programa e versão.
Acrescentar um método para mostrar valores pares em ordem crescente.
Na parte principal, editar a chamada do método para o novo.
Prever novos testes.

```
/**
  Method03a - Mostrar certa quantidade de valores pares.
  @param x - quantidade de valores a serem mostrados
 */
void method03a ( int x )
{
  // definir dado local
  int y = 1;           // controle
  int z = 0;

  // repetir enquanto controle menor que a quantidade desejada
  while ( y <= x )
  {
    // vincular o valor a ser mostrado ao controle
    z = 2 * y;
    // mostrar valor
    IO_printf ( "%d: %d\n", y, z );
    // passar ao proximo valor controlado
    y = y + 1;
  } // fim se
} // fim method03a( )

/**
  Method03.
 */
void method03 ( )
{
  // identificar
  IO_id ( "EXEMPLO0503 - Method03 - v0.0" );

  // executar o metodo auxiliar
  method03a ( 5 );

  // encerrar
  IO_pause ( "Apertar ENTER para continuar" );
} // fim method03 ( )
```

- 10.) Compilar o programa novamente.
Se houver erros, resolvê-los e compilar novamente, até que todos tenham sido resolvidos.
Se não houver erros, seguir para o próximo passo.
- 11.) Executar o programa. Observar as saídas. Registrar os dados e os resultados.
- 12.) Copiar a versão atual do programa para outra nova – Exemplo0504.c.
- 13.) Editar mudanças no nome do programa e versão.
Acrescentar um método para mostrar valores inteiros crescentes.
Na parte principal, editar a chamada do método para o novo.
Prever novos testes.

```
/**
    Method04a - Mostrar certa quantidade de valores pares.
    @param x - quantidade de valores a serem mostrados
*/
void method04a ( int x )
{
    // definir dado local
    int y = x;           // controle
    int z = 0;

    // repetir enquanto controle menor que a quantidade desejada
    while ( y > 0 )
    {
        // vincular o valor a ser mostrado ao controle
        z = 2 * y;
        // mostrar valor
        IO_printf ( "%d: %d\n", y, z );
        // passar ao proximo valor controlado
        y = y - 1;
    } // fim se
} // fim method04a ( )

/**
    Method04.
*/
void method04 ( )
{
    // identificar
    IO_id ( "EXEMPLO0504 - Method04 - v0.0" );

    // executar o metodo auxiliar
    method04a ( 5 );

    // encerrar
    IO_pause ( "Apertar ENTER para continuar" );
} // fim method04 ( )
```


- 14.) Compilar o programa novamente.
Se houver erros, resolvê-los e compilar novamente, até que todos tenham sido resolvidos.
Se não houver erros, seguir para o próximo passo.
- 15.) Executar o programa. Observar as saídas. Registrar os dados e os resultados.
- 16.) Copiar a versão atual do programa para outra nova – Exemplo0505.c.
- 17.) Editar mudanças no nome do programa e versão.
Acrescentar outro método para mostrar valores pares em ordem decrescente.
Na parte principal, editar a chamada do método para o novo.
Prever novos testes.

```
/**
  Method05a - Mostrar certa quantidade de valores pares.
  @param x - quantidade de valores a serem mostrados
 */
void method05a ( int x )
{
  // definir dado local
  int y = 0;           // controle

  // repetir enquanto controle menor que a quantidade desejada
  for ( y = x; y > 0; y = y-1 )
  {
    // mostrar valor
    IO_printf ( "%d: %d\n", y, (2*y) );
  } // fim se
} // fim method05a( )

/**
  Method05.
 */
void method05 ( )
{
  // identificar
  IO_id ( "EXEMPLO0505 - Method05 - v0.0" );

  // executar o metodo auxiliar
  method05a ( 5 );

  // encerrar
  IO_pause ( "Apertar ENTER para continuar" );
} // fim method05 ( )

/**
```

- 18.) Compilar o programa novamente.
Se houver erros, resolvê-los e compilar novamente, até que todos tenham sido resolvidos.
Se não houver erros, seguir para o próximo passo.
- 19.) Executar o programa. Observar as saídas. Registrar os dados e os resultados.

20.) Copiar a versão atual do programa para outra nova – Exemplo0506.c.

21.) Editar mudanças no nome do programa e versão.

Acrescentar uma função para somar valores da sequência: 1 + 2 + 4 + 6 + 8 ...

Na parte principal, editar a chamada do método para testar a função.

Prever novos testes.

```
/**
 * somarValores - funcao para somar certa quantidade de pares.
 * @return soma dos valores
 * @param x - quantidade de valores a serem mostrados
 */
int somarValores ( int x )
{
    // definir dados locais
    int soma = 0;
    int y = 0;           // controle

    // repetir enquanto controle menor que a quantidade desejada
    for ( y = x; y > 0; y = y-1 )
    {
        // mostrar valor
        IO_printf ( "%d: %d\n", y, (2*y) );
        // somar valor
        soma = soma + (2*y);
    } // fim se
    // retornar resultado
    return ( soma );
} // fim somarValores ( )

/**
 * Method06.
 */
void method06 ( )
{
    // definir dado
    int soma = 0;

    // identificar
    IO_id ( "EXEMPLO0506 - Method06 - v0.0" );

    // chamar e receber resultado da funcao
    soma = somarValores ( 5 );

    // mostrar resultado
    IO_printf ( "soma de pares = %d\n", soma );

    // encerrar
    IO_pause ( "Apertar ENTER para continuar" );
} // fim method06 ( )
```

OBS.: Reparar que a repetição se encerrará para o valor igual a x.
A repetição será feita, portanto, (x-1) vezes.

- 22.) Compilar o programa novamente.
Se houver erros, resolvê-los e compilar novamente, até que todos tenham sido resolvidos.
Se não houver erros, seguir para o próximo passo.
- 23.) Executar o programa.
Observar as saídas.
Registrar os dados e os resultados.
- 24.) Copiar a versão atual do programa para outra nova – Exemplo0507.c.
- 25.) Editar mudanças no nome do programa e versão.
Acrescentar outra função para somar valores na sequência:
 $1/1 + 1/2 + 1/4 + 1/6 + 1/8 \dots$
Na parte principal, editar a chamada do método para testar a função.
Prever novos testes.

```
/**
    somarFracao - funcao para somar certa quantidade de fracoes.
    @return soma dos valores
    @param x - quantidade de valores a serem mostrados
*/
double somarFracao ( int x )
{
    // definir dados locais
    double soma = 0.0;
    double numerador = 0;
    double denominador = 0;
    int y = 0 ;           // controle

    // repetir enquanto controle menor que a quantidade desejada
    for ( y = x; y > 0; y = y-1 )
    {
        // calcular numerador
        numerador = 1.0;
        // calcular denominador
        denominador = 2.0*y;
        // mostrar valor
        IO_printf ( "%d: %7.4lf/%7.4lf = %lf\n",
                    y, numerador, denominador, (numerador/denominador) );
        // somar valor
        soma = soma + (1.0)/(2.0*y);
    } // fim se
    // retornar resultado
    return ( soma );
} // fim somarFracao ( )
```

```

/**
  Method07.
 */
void method07 ( )
{
  // definir dado
  double soma = 0.0;

  // identificar
  IO_id ( "EXEMPLO0507 - Method07 - v0.0" );

  // chamar e receber resultado da funcao
  soma = somarFracao ( 5 );

  // mostrar resultado
  IO_printf ( "soma de fracoes = %lf\n", soma );

  // encerrar
  IO_pause ( "Apertar ENTER para continuar" );
} // fim method07 ( )

```

OBS.: Reparar que a repetição se iniciará para o valor igual a 2, e terminará em (x).
A repetição será feita, portanto, (x-1) vezes.

- 26.) Compilar o programa novamente.
Se houver erros, resolvê-los e compilar novamente, até que todos tenham sido resolvidos.
Se não houver erros, seguir para o próximo passo.
- 27.) Executar o programa.
Observar as saídas.
Registrar os dados e os resultados.
- 28.) Copiar a versão atual do programa para outra nova – Exemplo0508.c.
- 29.) Editar mudanças no nome do programa e versão.
Acrescentar outra função para somar os valores na sequência:
 $1/1 + 1/2 + 3/4 + 5/6 + 7/8 \dots$
Na parte principal, editar a chamada do método para testar a função.
Prever novos testes.

```

/**
    somarFracao - funcao para somar certa quantidade de fracoes.
    @return soma dos valores
    @param x - quantidade de valores a serem mostrados
*/
double somarFracao ( int x )
{
    // definir dados locais
    double soma = 1.0;
    double numerador = 0;
    double denominador = 0;
    int y = 0 ;           // controle

    // mostrar primeiro valor
    IO_printf ( "%d: %7.4lf/%7.4lf\n", 1, 1.0, soma );

    // repetir enquanto controle menor que a quantidade desejada
    for ( y = 1; y <= (x-1); y = y+1 )
    {
        // calcular numerador
        numerador = 2.0*y-1;
        // calcular denominador
        denominador = 2.0*y;
        // mostrar valor
        IO_printf ( "%d: %7.4lf/%7.4lf = %lf\n",
                    y, numerador, denominador, (numerador/denominador) );
        // somar valor
        soma = soma + (2.0*y-1)/(2.0*y);
    } // fim se
    // retornar resultado
    return ( soma );
} // fim somarFracao ( )

/**
    Method08.
*/
void method08 ( )
{
    // definir dado
    double soma = 0.0;

    // identificar
    IO_id ( "EXEMPLO0508 - Method08 - v0.0" );

    // chamar e receber resultado da funcao
    soma = somarFracao ( 5 );

    // mostrar resultado
    IO_printf ( "soma de fracoes = %lf\n", soma );

    // encerrar
    IO_pause ( "Apertar ENTER para continuar" );
} // fim method08 ( )

```

OBS.: Reparar que o valor inicial da soma será 1.0,
e um valor a menos da quantidade de vezes descontado.

- 30.) Compilar o programa novamente.
Se houver erros, resolvê-los e compilar novamente, até que todos tenham sido resolvidos.
Se não houver erros, seguir para o próximo passo.
- 31.) Executar o programa. Observar as saídas. Registrar os dados e os resultados.
- 32.) Copiar a versão atual do programa para outra nova – Exemplo0509.c.
- 33.) Editar mudanças no nome do programa e versão.
Acrescentar uma função para calcular a soma dos valores na sequência:
 $1/1 + 2/3 + 4/5 + 6/7 + 8/9 \dots$
Na parte principal, editar a chamada do método para o novo.
Prever novos testes.

```
/**
    somarFracao2 - funcao para somar certa quantidade de fracoes.
    @return soma dos valores
    @param x - quantidade de valores a serem mostrados
*/
double somarFracao2 ( int x )
{
    // definir dados locais
    double soma = 1.0;
    int y      = 0 ;           // controle

    // mostrar primeiro valor
    IO_printf ( "%d: %7.4lf/%7.4lf\n", 1, 1.0, soma );

    // repetir enquanto controle menor que a quantidade desejada
    for ( y = 1; y <= (x-1); y = y+1 )
    {
        // mostrar valor
        IO_printf ( "%d: %7.4lf/%7.4lf = %7.4lf\n",
                    y, (2.0*y+1), (2.0*y), ((2.0*y+1)/(2.0*y)) );

        // somar valor
        soma = soma + (2.0*y+1)/(2.0*y);
    } // fim se
    // retornar resultado
    return ( soma );
} // fim somarFracao2 ( )
```

```

/**
  Method09.
 */
void method09 ( )
{
  // definir dado
  double soma = 0.0;

  // identificar
  IO_id ( "EXEMPLO0509 - Method09 - v0.0" );

  // chamar e receber resultado da funcao
  soma = somarFracao2 ( 5 );

  // mostrar resultado
  IO_printf ( "soma de fracoes = %lf\n", soma );

  // encerrar
  IO_pause ( "Apertar ENTER para continuar" );
} // fim method09 ( )

```

- 34.) Compilar o programa novamente.
Se houver erros, resolvê-los e compilar novamente, até que todos tenham sido resolvidos.
Se não houver erros, seguir para o próximo passo.
- 35.) Executar o programa.
Observar as saídas.
Registrar os dados e os resultados.
- 36.) Copiar a versão atual do programa para outra nova – Exemplo0510.c.
- 37.) Editar mudanças no nome do programa e versão.
Acrescentar uma função para calcular o produto de valores na sequência:
1 * 3 * 5 * 7 * 9 ...
Na parte principal, editar a chamada do método para testar a função.
Prever novos testes.

```

/**
    multiplicarValores - funcao para multiplicar certa quantidade de valores pares.
    @return produto de valores
    @param x - quantidade de valores
*/
int multiplicarValores ( int x )
{
    // definir dados locais
    int produto = 1;
    int y      = 0;           // controle

    // repetir enquanto controle menor que a quantidade desejada
    for ( y = 1; y <= x; y = y+1 )
    {
        // mostrar valor
        IO_printf ( "%d: %d\n", y, (2*y-1) );
        // somar valor
        produto = produto * (2*y-1);
    } // fim se
    // retornar resultado
    return ( produto );
} // fim multiplicarValores ( )

/**
    Method10.
*/
void method10 ( )
{
    // identificar
    IO_id ( "EXEMPLO0510 - Method10 - v0.0" );

    // mostrar produto de valores
    IO_printf ( "%s%d\n", "produto = ", multiplicarValores ( 5 ) );

    // encerrar
    IO_pause ( "Apertar ENTER para continuar" );
} // fim method10 ( )

```

38.) Compilar o programa novamente.

Se houver erros, resolvê-los e compilar novamente, até que todos tenham sido resolvidos.

Se não houver erros, seguir para o próximo passo.

39.) Executar o programa. Observar as saídas. Registrar os dados e os resultados.

Exercícios:

DICAS GERAIS: Consultar o Anexo C 02 na apostila para outros exemplos.

Montar todos os métodos em um único programa conforme o último exemplo.

Incluir ao final desse programa os valores usados para testes.

- 01.) Incluir função e método em um programa (Exemplo0511) para:
para ler uma quantidade inteira do teclado e, mediante um método,
mostrar essa quantidade em valores múltiplos de 2 em ordem crescente.
- 02.) Incluir função e método em um programa (Exemplo0512) para:
para ler uma quantidade inteira do teclado e, mediante um método
mostrar essa quantidade em valores pares múltiplos de 2 em ordem crescente.
- 03.) Incluir função e método em um programa (Exemplo0513) para:
para ler uma quantidade inteira do teclado e, mediante um método,
mostrar essa quantidade em valores ímpares múltiplos de 2 em ordem decrescente.
- 04.) Incluir função e método em um programa (Exemplo0514) para:
para ler uma quantidade inteira do teclado e, mediante um método,
mostrar essa quantidade em valores crescentes nos denominadores
(sequência dos inversos) múltiplos de 3: $1 \frac{1}{2} \frac{1}{4} \frac{1}{6} \frac{1}{8} \frac{1}{10} \dots$
- 05.) Incluir função e método em um programa (Exemplo0515) para:
para ler um valor real (x) do teclado;
para ler uma quantidade inteira do teclado e, mediante um método,
mostrar essa quantidade em valores crescentes nos denominadores
da sequência: $1 \frac{1}{x} \frac{1}{x^2} \frac{1}{x^3} \dots$
DICA: Usar da biblioteca <math.h> a função **pow (x, y)** para calcular a potência.
- 06.) Incluir função e método em um programa (Exemplo0516) para
calcular a soma dos primeiros valores pares positivos começando em 2
e não múltiplos de 5.
Testar essa função para quantidades diferentes.
- 07.) Incluir função e método em um programa (Exemplo0517) para
calcular a soma dos inversos ($1/x$) dos primeiros valores pares positivos,
começando em 2 e não múltiplos de 5.
Testar essa função para quantidades diferentes.
- 08.) Incluir função e método em um programa (Exemplo0518) para
calcular a soma dos primeiros números naturais começando em 1.
Testar essa função para quantidades diferentes de valores.

09.) Incluir função e método em um programa (Exemplo0518) para calcular a soma dos quadrados dos primeiros números naturais começando em 1. Testar essa função para quantidades diferentes de valores.

10.) Incluir função e método em um programa (Exemplo0518) para calcular a soma dos inversos dos primeiros números naturais começando em 1. Testar essa função para quantidades diferentes de valores.

Tarefas extras

E1.) Incluir função e método em um programa (Exemplo05E1) para ler um número inteiro do teclado e, mediante o uso da função, calcular e mostrar o fatorial desse valor:

$$n! = n * (n-1) * (n-2) * \dots * 3 * 2 * 1 \quad \text{se } n > 0$$

E2.) Incluir função e método em um programa (Exemplo05E2) para ler uma quantidade inteira do teclado e, mediante o uso da função, calcular e mostrar o resultado de

$$f(n) = (1 + 2/3!) * (1 + 3/4!) * (1 + 4/5!) * \dots$$