PUC-Minas - Ciência da Computação
AED1 – Estudo Dirigido 05

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 Exemplo0500.c, com método para ler e mostrar certa quantidade de valores:

```
/**
   Method_01a - Mostrar certa quantidade de valores.
   @param x - quantidade de valores a serem mostrados
 */
void method_01a ( 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;
    } // end if
} // end method_01a( )

/**
   Method_01 - Mostrar certa quantidade de valores.
          OBS.:   Preparacao e disparo de outro metodo.
 */
void method_01 ( )
{
 // identificar
    IO_id ( " Method_01 - v0.0" );

 // executar o metodo auxiliar
    method_01a ( 5 );

 // encerrar
    IO_pause ( "Apertar ENTER para continuar" );
} // end method_01 ( )
```

```
/*
---------------------------------------------- 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.) 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.

```
/**
  Method_02a - Mostrar certa quantidade de valores pares.
  @param x - quantidade de valores a serem mostrados
 */
void method_02a ( 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;
   } // end while
} // end method_02a( )
```

```
/**
  Method_02.
 */
void method_02 ( )
{
 // identificar
    IO_id ( "Method_02 - v0.0" );

 // executar o metodo auxiliar
    method_02a ( 5 );

 // encerrar
    IO_pause ( "Apertar ENTER para continuar" );
} // end method_02 ( )
```

05.) 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.

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

07.) Acrescentar um método para mostrar valores pares em ordem crescente.
Na parte principal, editar a chamada do método para o novo.
Prever novos testes.

```
/**
  Method_03a - Mostrar certa quantidade de valores pares.
  @param x - quantidade de valores a serem mostrados
 */
void method_03a ( 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;
    } // end while
} // end method_03a( )
```

```
/**
  Method_03.
 */
void method_03 ( )
{
 // identificar
   IO_id ( " Method_03 - v0.0" );

 // executar o metodo auxiliar
   method_03a ( 5 );

 // encerrar
   IO_pause ( "Apertar ENTER para continuar" );
} // end method_03 ( )
```

08.) 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.

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

10.) Acrescentar um método para mostrar valores inteiros crescentes.
Na parte principal, editar a chamada do método para o novo.
Prever novos testes.

```
/**
  Method_04a - Mostrar certa quantidade de valores pares.
  @param x - quantidade de valores a serem mostrados
 */
void method_04a ( 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;
   } // end while
} // end method_04a( )
```

```
/**
  Method_04.
 */
void method_04 ( )
{
 // identificar
   IO_id ( "Method_04 - v0.0" );

 // executar o metodo auxiliar
   method_04a ( 5 );

 // encerrar
   IO_pause ( "Apertar ENTER para continuar" );
} // end method_04 ( )
```

11.) 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.

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

13.) Acrescentar outro método para mostrar valores pares em ordem decrescente.
Na parte principal, editar a chamada do método para o novo.
Prever novos testes.

```
/**
  Method_05a - Mostrar certa quantidade de valores pares.
  @param x - quantidade de valores a serem mostrados
 */
void method_05a ( 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) );
   } // end for
} // end method_05a( )
```

```
/**
  Method_05.
 */
void method_05 ( )
{
 // identificar
    IO_id ( "Method_05 - v0.0" );

 // executar o metodo auxiliar
    method_05a ( 5 );

 // encerrar
    IO_pause ( "Apertar ENTER para continuar" );
} // end method_05 ( )

/**
```

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.) 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 = 1;
    int y       = 0;                    // controle

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

```
/**
  Method_06.
 */
void method_06 ( )
{
 // definir dado
    int soma = 0;

 // identificar
    IO_id ( "Method_06 - 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" );
} // end method_06 ( )
```

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

17.) 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.

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

19.) Acrescentar outra função para somar valores na sequência:
1/1 + 1/2 + 1/4 + 1/6 + 1/8 ...
Na parte principal, editar a chamada do método para testar a função.
Prever novos testes.

```
/**
  somarFracao1 - funcao para somar certa quantidade de fracoes.
  @return soma dos valores
  @param x - quantidade de valores a serem mostrados
 */
double somarFracao1 ( int x )
{
 // definir dados locais
    double soma           = 1.0;
    double numerador    = 0.0;
    double denominador = 0.0;
    int y                       =  0  ;                 // controle
```

```
 // repetir enquanto controle menor que a quantidade desejada
    for ( y = 1; y <= (x-1); 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);
    } // end for
 // retornar resultado
    return ( soma );
} // end somarFracao1 ( )

/**
   Method_07.
 */
void method_07 ( )
{
 // definir dado
    double soma = 0.0;

 // identificar
    IO_id ( "Method_07 - v0.0" );

 // chamar e receber resultado da funcao
    soma = somarFracao1 ( 5 );

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

 // encerrar
    IO_pause ( "Apertar ENTER para continuar" );
} // end method_07 ( )
```

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.

20.) 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.

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

22.) Acrescentar outra função para somar os valores na sequência:
1/1 + 1/2 + 3/4 + 5/6 + 7/8 ...
Na parte principal, editar a chamada do método para testar a função.
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;
   double numerador     = 0.0;
   double denominador = 0.0;
   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+1, numerador, denominador, (numerador/denominador) );
    // somar valor
      soma = soma + numerador / denominador;
   } // end for
 // retornar resultado
   return ( soma );
} // end somarFracao2 ( )

/**
  Method_08.
 */
void method_08 ( )
{
 // definir dado
   double soma = 0.0;

 // identificar
   IO_id ( "Method_08 - 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" );
} // end method_08 ( )
```

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

23.) 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.

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

25.) Acrescentar uma função para calcular a soma dos valores na sequência:
1/1 + 2/3 + 4/5 + 6/7 + 8/9 ...
Na parte principal, editar a chamada do método para o novo.
Prever novos testes.

```
/**
  somarFracao3 - funcao para somar certa quantidade de fracoes.
  @return soma dos valores
  @param x - quantidade de valores a serem mostrados
 */
double somarFracao3 ( 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; y = y+1 )
   {
    // mostrar valor
      IO_printf ( "%d: %7.4lf/%7.4lf = %7.4lf\n",
                  y+1, (2.0*y), (2.0*y+1), ((2.0*y)/(2.0*y+1)) );
    // somar valor
      soma = soma + (2.0*y)/(2.0*y+1);
   } // end for
 // retornar resultado
   return ( soma );
} // end somarFracao3 ( )
```

```
/**
  Method_09.
 */
void method_09 ( )
{
 // definir dado
   double soma = 0.0;

 // identificar
   IO_id ( "EXEMPLO0509 - Method_09 - v0.0" );

 // chamar e receber resultado da funcao
   soma = somarFracao3 ( 5 );

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

 // encerrar
   IO_pause ( "Apertar ENTER para continuar" );
} // end method_09 ( )
```

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.) 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 impares.
  @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);
   } // end for
 // retornar resultado
   return ( produto );
} // end multiplicarValores ( )
```

```
/**
  Method_10.
 */
void method_10 ( )
{
 // identificar
   IO_id ( "Method_10 - v0.0" );

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

 // encerrar
   IO_pause ( "Apertar ENTER para continuar" );
} // end method_10 ( )
```

29.) 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.

30.) 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.
Os métodos deverão preparar dados, chamar funções e mostrar resultados.
Incluir ao final desse programa os dados e os resultados de testes.

01.) Incluir função e método (0511) para:
para ler uma quantidade inteira (n) do teclado e, mediante um procedimento,
mostrar essa quantidade em valores múltiplos de 7 em ordem crescente.

Exemplo: n = 5 => { 7, 14, 21, 28, 35 }

02.) Incluir função e método (0512) para:
para ler uma quantidade inteira do teclado e, mediante um procedimento,
mostrar essa quantidade em valores múltiplos de 3 e 4 em ordem crescente.

Exemplo: n = 5 => { 12, 24, 36, 48, 60 }

03.) Incluir função e método (0513) para:
para ler uma quantidade inteira do teclado e, mediante um procedimento,
mostrar essa quantidade em valores ímpares potências de 5 em ordem decrescente.

Exemplo: n = 5 => { 3125, 625, 125, 25, 5 }

04.) Incluir função e método (0514) para:
para ler uma quantidade inteira do teclado e, mediante um procedimento,
mostrar essa quantidade em valores crescentes nos denominadores
(sequência dos inversos) múltiplos de 6.

Exemplo: n = 5 => { 1/7, 1/14, 1/21, 1/28, 1/35 }

05.) Incluir função e método (0515) para:
para ler um valor real real (x) do teclado;
para ler uma quantidade inteira do teclado e, mediante um procedimento,
mostrar essa quantidade em valores ímpares crescentes nos denominadores
da sequência: 1 1/x $1/x^3$ $1/x^5$ ...
DICA: Usar da biblioteca <math.h> a função **pow (x, y)** para calcular a potência.

Exemplo: n = 5 => { 1, 1/x, $1/x^3$,$1/x^5$, $1/x^7$ }

06.) Incluir função e método (0516) para
calcular a soma dos primeiros valores pares positivos começando no valor 3
e não múltiplos de 4.
Testar essa função para quantidades diferentes e mostrar os resultados em outro método.

Exemplo: n = 5 => 3 + 6 + 9 + 15 + 18

07.) Incluir função e método (0517) para
calcular a soma dos inversos (1/x) dos primeiros valores pares positivos,
começando no valor 6 e não múltiplos de 5.
Testar essa função para quantidades diferentes e
mostrar os resultados em outro método.

Exemplo: n = 5 => 1/7 + 1/14 + 1/21 + 1/28 + 1/42

08.) Incluir função e método (0518) para
calcular a soma da adição dos primeiros números naturais começando no valor 7.
Testar essa função para quantidades diferentes de valores e
mostrar os resultados em outro método.

Exemplo: n = 5 => 7 + 8 + 10 + 13 + 17

09.) Incluir função e método (0519) para
calcular a soma dos quadrados da adição dos números naturais começando no valor 7.
Testar essa função para quantidades diferentes de valores e
e mostrar os resultados em outro método.

Exemplo: n = 5 => 49 + 81 + 144 + 256 + 289

10.) Incluir função e método (Exemplo0520) para
calcular a soma dos inversos (1/x) das adições de números naturais começando no valor 6.
Testar essa função para quantidades diferentes de valores
e mostrar os resultados em outro método.

Exemplo: n = 5 => 1/6 + 1/7 + 1/9 + 1/12 + 1/16

Tarefas extras

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

n! = n * (n-1) * (n-2) * ... * 3 * 2 * 1 se n>0

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

f (n) = (1+3/5!) * (1+4/7!) * (1+5/9!) * ...