PUC-Minas - Ciência da Computação
AED1 – Estudo Dirigido 06

Tema: Introdução à programação II
Atividade: Funções e procedimentos recursivos em C

01.) Editar e salvar um esboço de programa em C, cujo nome será Exemplo0600.c,
com método para mostrar certa quantidade de valores recursivamente:

```
/**
  Method_01a - Mostrar certa quantidade de valores recursivamente.
  @param x - quantidade de valores a serem mostrados
 */
void method_01a ( int x )
{
 // repetir enquanto valor maior que zero
   if ( x > 0 )
   {
    // mostrar valor
      IO_printf ( "%s%d\n", "Valor = ", x );
    // passar ao proximo
      method_01a ( x - 1 );          // motor da recursividade
   } // end if
} // end method_01a( )

/**
  Method_01 - Mostrar certa quantidade de valores.
 */
void method_01 ( )
{
 // definir dado
   int quantidade = 0;
   int valor      = 0;
   int controle   = 0;

 // identificar
   IO_id ( "Method_01 - v0.0" );

 // executar o metodo auxiliar
   method_01a ( 5 );              // motor da recursividade

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

02.) Montar a função principal e 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.

04.) Acrescentar um método para mostrar certa quantidade de valores positivos.
Na parte principal, incluir a chamada de um método para testar o novo.
Prever novos testes.

```
/**
   Method_02a - Mostrar certa quantidade de valores recursivamente.
   @param x - quantidade de valores a serem mostrados
 */
void method_02a ( int x )
{
 // repetir enquanto valor maior que zero
    if ( x > 0 )
    {
     // passar ao proximo
        method_02a ( x - 1 );           // motor da recursividade
     // mostrar valor
        IO_printf ( "%s%d\n", "Valor = ", x );
    } // end if
} // end method_02a( )

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

 // executar o metodo auxiliar
    method_02a ( 5 );                   // motor da recursividade

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

OBS.:
A exibição do primeiro valor não ocorrerá enquanto
o parâmetro (x) não chegar a zero, e não for iniciar o processo de retorno.
Os valores pendentes serão conhecidos durante o retorno.

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 recursivo para mostrar certa quantidade de valores positivos, em outra ordem.
Na parte principal, incluir a chamada de um método para testar o novo.
Prever novos testes.

```
/**
  Method_03a - Mostrar certa quantidade de valores recursivamente.
  @param x - quantidade de valores a serem mostrados
 */
void method_03a ( int x )
{
 // repetir enquanto valor maior que zero
   if ( x > 1 )
   {
    // passar ao proximo
      method_03a ( x - 1 );          // motor da recursividade
    // mostrar valor
      IO_printf ( "%s%d\n", "Valor = ", x );
   }
   else
   {                                 // base  da recursividade
    // mostrar o ultimo
      IO_printf ( "%s\n", "Valor = 1" );
   } // end if
} // end method_03a( )

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

 // executar o metodo auxiliar
   method_03a ( 5 );                 // motor da recursividade

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

OBS.:
Diferente do anterior, a exibição do primeiro valor ocorrerá antes de avançar para o próximo valor (motor).
Observar também que o último valor será tratado de forma particular (base).

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 outro método para mostrar valores da sequência: 1 2 4 6 8...
Na parte principal, incluir a chamada de um método para testar o novo.
Prever novos testes.

```
/**
  Method_04a - Mostrar certa quantidade de valores recursivamente.
  @param x - quantidade de valores a serem mostrados
 */
void method_04a ( int x )
{
 // repetir enquanto valor maior que zero
   if ( x > 1 )
   {
    // passar ao proximo
      method_04a ( x - 1 );          // motor da recursividade
    // mostrar valor
      IO_printf ( "%s%d\n", "Valor = ", 2*(x-1) );
   }
   else
   {                                 // base  da recursividade
    // mostrar o ultimo
      IO_printf ( "%s\n", "Valor = 1" );
   } // end if
} // end method_04a( )

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

 // executar o metodo auxiliar
   method_04a ( 5 );                 // motor da recursividade

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

OBS.:
Observar que o último valor será tratado de forma particular.

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 de parcelas do somatório:
1 + 2/3 + 4/5 + 6/7 + 8/9 ...
Na parte principal, incluir a chamada de um método para testar o novo.
Prever novos testes.

```
/**
  Method_05a - Mostrar certa quantidade de valores recursivamente.
  @param x - quantidade de valores a serem mostrados
 */
void method_05a ( int x )
{
 // repetir enquanto valor maior que zero
   if ( x > 1 )
   {
    // passar ao proximo
      method_05a ( x - 1 );          // motor da recursividade
    // mostrar valor
      IO_printf ( "%d: %d/%d\n", x, (2*(x-1)), (2*(x-1)+1) );
   }
   else
   {                                 // base  da recursividade
    // mostrar o ultimo
      IO_printf ( "%d; %d\n", x, 1 );
   } // end if
} // end method_05a( )

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

 // executar o metodo auxiliar
   method_05a ( 5 );                 // motor da recursividade

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

OBS.:
Observar que o primeiro na sequência será tratado de forma particular.

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 calcular o somatório: 1 + 2/3 + 4/5 + 6/7 + ...
Na parte principal, incluir a chamada de um método para testar essa função.
Prever novos testes.

```
/**
  somarFracoes - Somar certa quantidade de fracoes recursivamente.
  @return soma de valores
  @param x - quantidade de valores a serem mostrados
 */
double somarFracoes ( int x )
{
 // definir dado local
   double soma = 0.0;

 // repetir enquanto valor maior que zero
   if ( x > 1 )
   {
    // separar um valor e passar ao proximo (motor da recursividade)
      soma = (2.0*(x-1))/(2.0*(x-1)+1) + somarFracoes ( x - 1 );
    // mostrar valor
      IO_printf ( "%d: %lf/%lf\n", x, (2.0*(x-1)), (2.0*(x-1)+1) );
   }
   else
   {
    // base  da recursividade
      soma = 1.0;
    // mostrar o ultimo
      IO_printf ( "%d; %lf\n", x, 1.0 );
   } // end if
 // retornar resultado
   return ( soma );
} // end somarFracoes ( )

/**
  Method_06.
 */
void method_06 ( )
{
 // definir dado
   double soma = 0.0;

 // identificar
   IO_id ( "Method_06 - v0.0" );

 // chamar a funcao e receber o resultado
   soma = somarFracoes ( 5 );

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

 // encerrar
   IO_pause ( "Apertar ENTER para continuar" );
} // end method_06 ( )
```

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 uma função para calcular o somatório: 1 + 2/3 + 4/5 + 6/7 + 8/9 ...
Na parte principal, incluir a chamada de um método para testar essa função.
Prever novos testes.

```
/**
   somarFracoes2b - Somar certa quantidade de fracoes recursivamente.
   @return soma de valores
   @param x                - quantidade de valores a serem mostrados (controle)
   @param soma             - valor atual da soma (historia = memoria)
   @param numerador    - numerador     da parcela a ser somada
   @param denominador - denominador da parcela a ser somada
 */
double somarFracoes2b ( int x, double soma, double numerador, double denominador )
{
 // repetir enquanto valor maior que zero
    if ( x > 0 )
    {
     // mostrar valores atuais
        IO_printf ( "%d: %lf/%lf\n", x, numerador, denominador );
     // somar o termo atual e passar ao proximo (motor da recursividade)
        soma = somarFracoes2b ( x - 1,                                          // proximo
                                soma + ((1.0*numerador) / denominador), // atualizar
                                numerador    +2.0  ,                          // proximo
                                denominador+2.0  );                          // proximo
    } end if

 // retornar resultado
    return ( soma );
} // end somarFracoes2b ( )

/**
   somarFracoes2a - Somar certa quantidade de fracoes.
                    Funcao de servico para preparar e
                    disparar o mecanismo recursivo.
   @return soma de valores
   @param x - quantidade de valores a serem mostrados
 */
double somarFracoes2a ( int x )
{
 // definir dado local
    double soma = 0.0;

 // repetir enquanto valor maior que zero
    if ( x > 0 )
    {
     // mostrar o ultimo
        IO_printf ( "%d: %lf\n", x, 1.0 );
     // preparar a soma do valor atual e o proximo
        soma = somarFracoes2b ( x-1, 1.0, 2.0, 3.0 );
    } // end if

 // retornar resultado
    return ( soma );
} // end somarFracoes2a ( )
```

```
/**
  Method_07.
 */
void method_07 ( )
{
 // definir dado
    double soma = 0.0;
 // identificar
    IO_id ( "Method_07 - v0.0" );
 // chamar a funcao e receber o resultado
    soma = somarFracoes2a ( 5 );
 // mostrar resultado
    IO_printf ( "soma = %lf\n", soma );
 // encerrar
    IO_pause ( "Apertar ENTER para continuar" );
} // end method_07 ( )
```

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 uma função para calcular a quantidade de dígitos de um valor inteiro.
Na parte principal, incluir chamada a um método para testar essa função.
Prever novos testes.

```
/**
  contarDigitos - Contar digitos recursivamente.
  @return quantidade de digitos
  @param x - numero cuja quantidade de digitos sera' calculada
 */
int contarDigitos ( int x )
{
 // definir dado
    int resposta = 1;                          // base

 // testar se contador valido
    if ( x >= 10 )
    {
     // tentar fazer de novo com valor menor
        resposta = 1 + contarDigitos ( x/10 ); // motor 1
    }
    else
    {
        if ( x < 0 )
        {
         // fazer de novo com valor absoluto
            resposta = contarDigitos ( -x );       // motor 2
        } // end if
    } // end if
 // retornar resposta
    return ( resposta );
} // end contarDigitos ( )
```

```
/**
  Method_08.
 */
void method_08 ( )
{
 // identificar
   IO_id ( "Method_08 - v0.0" );

 // mostrar resultado
   IO_printf ( "digitos (%3d) = %d\n", 123, contarDigitos (123) );
   IO_printf ( "digitos (%3d) = %d\n",   1  , contarDigitos (  1  ) );
   IO_printf ( "digitos (%3d) = %d\n", -10, contarDigitos ( -10 ) );

 // encerrar
   IO_pause ( "Apertar ENTER para continuar" );
} // end method_08 ( )
```

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 termo da série de Fibonacci.
Na parte principal, incluir chamada ao método para testar essa função.
Prever novos testes.

```
/**
  fibonacci - Gerador de numero de Fibonacci.
  @return numero de Fibonacci
  @param x - numero de ordem cujo valor sera' calculado
 */
int fibonacci ( int x )
{
 // definir dado
   int resposta = 0;

 // testar se contador valido
   if ( x == 1 || x == 2 )
   {
    // primeiros dois valores iguais a 1
      resposta = 1;                    // bases
   }
   else
   {
      if ( x > 1 )
      {
       // fazer de novo com valor absoluto
          resposta = fibonacci ( x-1 ) + fibonacci ( x-2 );
      } // end if
   } // end if

 // retornar resposta
   return ( resposta );
} // end fibonacci ( )
```

```
/**
  Method_09.
 */
void method_09 ( )
{
 // identificar
   IO_id ( "Method_09 - v0.0" );

 // calcular numero de Fibonacci
   IO_printf ( "fibonacci (%d) = %d\n", 1, fibonacci ( 1 ) );
   IO_printf ( "fibonacci (%d) = %d\n", 2, fibonacci ( 2 ) );
   IO_printf ( "fibonacci (%d) = %d\n", 3, fibonacci ( 3 ) );
   IO_printf ( "fibonacci (%d) = %d\n", 4, fibonacci ( 4 ) );
   IO_printf ( "fibonacci (%d) = %d\n", 5, fibonacci ( 5 ) );

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

26.) Compilar o programa novamente. Se houver erros, resolvê-los; senão seguir para o próximo passo.

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

27.) Acrescentar uma função para contar letras minúsculas em uma cadeia de caracteres.
Na parte principal, incluir chamada ao método para testar essa função.
Prever novos testes.

```
/**
  contarMinusculas - Contador de letras minusculas.
  @return quantidade de letras minusculas
  @param x - cadeia de caracteres a ser avaliada
 */
int contarMinusculas ( chars cadeia, int x )
{
 // definir dado
   int resposta = 0;

 // testar se contador valido
   if ( 0 <= x && x < strlen ( cadeia ) )
   {
    // testar se letra minuscula
      if ( cadeia [x] >= 'a' &&
           cadeia [x] <= 'z' )
      {
       // fazer de novo com valor absoluto
          resposta = 1;
      } // end if
      resposta = resposta + contarMinusculas ( cadeia, x+1 );
   } // end if
 // retornar resposta
   return ( resposta );
} // end contarMinusculas ( )
```

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

 // contar minusculas em cadeias de caracteres
   IO_printf ( "Minusculas (\"abc\",0)  = %d\n", contarMinusculas ( "abc", 0 ) );
   IO_printf ( "Minusculas (\"aBc\",0)  = %d\n", contarMinusculas ( "aBc", 0 ) );
   IO_printf ( "Minusculas (\"AbC\",0) = %d\n", contarMinusculas ( "AbC", 0 ) );

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

Exercícios:

DICAS GERAIS: Consultar o Anexo C 02 na apostila para outros exemplos.
Prever, realizar e registrar todos os testes efetuados.
Integrar as chamadas de todos os programas em um só.

01.) Incluir um método recursivo (0611) para
ler um valor inteiro do teclado e chamar procedimento recursivo para
mostrar essa quantidade em valores múltiplos de 6, em ordem crescente, começando no valor 6.

Exemplo: valor = 5 => { 6, 12, 18, 24, 30 }

02.) Incluir um método recursivo (0612) para
ler um valor inteiro do teclado e chamar procedimento recursivo para
mostrar essa quantidade em múltiplos de 6, em ordem decrescente, encerrando no valor 6.

Exemplo: valor = 5 => { 30, 24, 18, 12, 6 }

03.) Incluir um método recursivo (0613) para
ler um valor inteiro do teclado e chamar procedimento recursivo para
mostrar essa quantidade em valores da sequência de inversos dos múltiplos de 6, começando em 1.
DICA: Separar o primeiro do restante.

Exemplo: valor = 5 => { 1/1, 1/6, 1/12, 1/18, 1/24. }

04.) Incluir um método recursivo (0614) para
ler um valor inteiro do teclado e chamar procedimento recursivo para
mostrar essa quantidade em valores da sequência de inversos dos múltiplos de 6, terminando em 1.

Exemplo: valor = 5 => { 1/24, 1/18, 1/12, 1/6, 1/1 }

05.) Incluir uma função recursiva (0615) para
calcular a soma de valores gerados pela adição dos primeiros valores pares positivos ao valor 6.
Testar essa função para quantidades diferentes.

Exemplo: valor = 5 => 6 + 8 + 12 + 18 + 26

06.) Incluir uma função recursiva (0616) para
calcular a soma dos inversos (1/x) dos primeiros valores pares positivos múltiplos do valor 6.
Testar essa função para quantidades diferentes.

Exemplo: valor = 5 => 1/6 + 1/12 + 1/18 + 1/24 + 1/30

07.) Incluir um método recursivo (0617) para
ler uma cadeia de caracteres e chamar procedimento recursivo para
mostrar cada símbolo separadamente, um por linha.

Exemplo: sequência = "abcde"

08.) Incluir uma função recursiva (0618) para
contar os dígitos com valores pares em uma cadeia de caracteres.
Testar essa função para cadeias de diferentes tamanhos.

Exemplo: sequência = "P4LaVr@1"

09.) Incluir uma função recursiva (0619) para
calcular a quantidade de maiúsculas menores que 'L' em uma cadeia de caracteres.
Testar essa função para cadeias de diferentes tamanhos.

Exemplo: sequência = "P4LaVr@1"

10.) Incluir uma função recursiva (0620) para
calcular certo termo par da série de Fibonacci começando em 1.
Testar essa função para quantidades diferentes.
DICA: Separar o cálculo do termo e o teste para verificar se é par.

Exemplo: valor = 3 => 2+8+34

Tarefas extras

E1.) Incluir uma função recursiva (06E1) para
calcular o valor da função definida abaixo,
lidos os valores de (x) e (n) do teclado:

$f (x, n) = 1 + x^3 + x^5 + x^7 + x^9 + ...$

E2.) Incluir uma função recursiva (06E2) para
para calcular o valor indicado abaixo,
lido o número de termos (n) do teclado:

$e = 1 + 3/2! + 5/4! + 7/6! + 9/8! + ...$