

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 Exemplo0601.c, para mostrar certa quantidade de valores recursivamente:

```
/**
 * Method01a - Mostrar certa quantidade de valores recursivamente.
 * @param x - quantidade de valores a serem mostrados
 */
void method01a ( int x )
{
    // repetir enquanto valor maior que zero
    if ( x > 0 )
    {
        // mostrar valor
        IO_printf ( "%s%d\n", "Valor = ", x );
        // passar ao proximo
        method01a ( x - 1 );          // motor da recursividade
    } // fim se
} // fim method01a ( )

/**
 * Method01 - Mostrar certa quantidade de valores.
 */
void method01 ( )
{
    // definir dado
    int quantidade = 0;
    int valor      = 0;
    int controle   = 0;

    // identificar
    IO_id ( "EXEMPLO0601 - Method01 - v0.0" );

    // executar o metodo auxiliar
    method01a ( 5 );                // motor da recursividade

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

- 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.) Copiar a versão atual do programa para outra nova – Exemplo0602.c.

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

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.

```
/**
 * Method02a - Mostrar certa quantidade de valores recursivamente.
 * @param x - quantidade de valores a serem mostrados
 */
void method02a ( int x )
{
    // repetir enquanto valor maior que zero
    if ( x > 0 )
    {
        // passar ao proximo
        method02a ( x - 1 );          // motor da recursividade
        // mostrar valor
        IO_printf ( "%s%d\n", "Valor = ", x );
    } // fim se
} // fim method02a( )

/**
 * Method02.
 */
void method02 ( )
{
    // identificar
    IO_id ( "EXEMPLO0602 - Method02 - v0.0" );

    // executar o metodo auxiliar
    method02a ( 5 );                // motor da recursividade

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

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.

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 – Exemplo0603.c.

- 09.) Editar mudanças no nome do programa e versão.
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.

```
/**
  Method03a - Mostrar certa quantidade de valores recursivamente.
  @param x - quantidade de valores a serem mostrados
 */
void method03a ( int x )
{
  // repetir enquanto valor maior que zero
  if ( x > 1 )
  {
    // passar ao proximo
    method03a ( 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" );
  } // fim se
} // fim method03a( )

/**
  Method03.
 */
void method03 ( )
{
  // identificar
  IO_id ( "EXEMPLO0603 - Method03 - v0.0" );

  // executar o metodo auxiliar
  method03a ( 5 );              // motor da recursividade

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

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).

- 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 – Exemplo0604.c.

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

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.

```
/**
 * Method04a - Mostrar certa quantidade de valores recursivamente.
 * @param x - quantidade de valores a serem mostrados
 */
void method04a ( int x )
{
    // repetir enquanto valor maior que zero
    if ( x > 1 )
    {
        // passar ao proximo
        method04a ( 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" );
    } // fim se
} // fim method04a ( )

/**
 * Method04.
 */
void method04 ( )
{
    // identificar
    IO_id ( "EXEMPLO0604 - Method04 - v0.0" );

    // executar o metodo auxiliar
    method04a ( 5 );                // motor da recursividade

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

OBS.:

Observar que o último valor 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.) Copiar a versão atual do programa para outra nova – Exemplo0605.c.

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

Acrescentar outro método para mostrar valores de parcelas do somatório:

$1 + 2/3 + 4/5 + 6/7 + 8/9 \dots$

Na parte principal, incluir a chamada de um método para testar o novo.

Prever novos testes.

```
/**
    Method05a - Mostrar certa quantidade de valores recursivamente.
    @param x - quantidade de valores a serem mostrados
 */
void method05a ( int x )
{
    // repetir enquanto valor maior que zero
    if ( x > 1 )
    {
        // passar ao proximo
        method05a ( 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 );
    } // fim se
} // fim method05a ( )

/**
    Method05.
 */
void method05 ( )
{
    // identificar
    IO_id ( "EXEMPLO0605 - Method05 - v0.0" );

    // executar o metodo auxiliar
    method05a ( 5 );                // motor da recursividade

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

OBS.:

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

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 – Exemplo0606.c.

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

Acrescentar uma função para calcular o somatório: $1 + 2/3 + 4/5 + 6/7 + \dots$

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 );
    } // fim se
    // retornar resultado
    return ( soma );
} // fim somarFracoes ( )

/**
    Method06.
*/
void method06 ( )
{
    // definir dado
    double soma = 0.0;

    // identificar
    IO_id ( "EXEMPLO0606 - Method06 - 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" );
} // fim method06 ( )
```

- 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 – Exemplo0607.c.
- 25.) Editar mudanças no nome do programa e versão.
 Acrescentar uma função para calcular o somatório: $1 + 2/3 + 4/5 + 6/7 + 8/9 \dots$
 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
}

// retornar resultado
return ( soma );
} // fim 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 );
    } // fim se

    // retornar resultado
    return ( soma );
} // fim somarFracoes2a ( )

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

    // identificar
    IO_id ( "EXEMPLO0607 - Method07 - 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" );
} // fim method07 ( )

```

- 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 – Exemplo0608.c.

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

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
        } // fim se
    } // fim se
    // retornar resposta
    return ( resposta );
} // fim contarDigitos ( )

/**
    Method08.
*/
void method08 ( )
{
    // identificar
    IO_id ( "EXEMPLO0608 - Method08 - 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" );
} // fim method08 ( )
```

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 – Exemplo0609.c.

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

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 );
        } // fim se
    } // fim se

    // retornar resposta
    return ( resposta );
} // fim fibonacci ( )

/**
    Method09.
*/
void method09 ( )
{
    // identificar
    IO_id ( "EXEMPLO0609 - Method09 - 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" );
} // fim method09 ( )
```

34.) Compilar o programa novamente. Se houver erros, resolvê-los; senão 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 – Exemplo0610.c.

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

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;
        } // fim se
        resposta = resposta + contarMinusculas ( cadeia, x+1 );
    } // fim se
    // retornar resposta
    return ( resposta );
} // fim contarMinusculas ( )

/**
    Method10.
*/
void method10 ( )
{
    // identificar
    IO_id ( "EXEMPLO0610 - Method10 - 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" );
} // fim method10 ( )
```

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 em um programa (Exemplo0611) um método para ler um valor inteiro do teclado e chamar procedimento recursivo para mostrar essa quantidade em valores pares em ordem crescente começando em 5.
- 02.) Incluir em um programa (Exemplo0612) um método para ler um valor inteiro do teclado e chamar procedimento recursivo para mostrar essa quantidade em múltiplos de 5 em ordem decrescente encerrando em 5.
- 03.) Incluir em um programa (Exemplo0613) um método para ler um valor inteiro do teclado e chamar procedimento recursivo para mostrar essa quantidade em valores da sequência: 1 3 6 9 12 15 ...
- 04.) Incluir em um programa (Exemplo0614) um método para ler um valor inteiro do teclado e chamar procedimento recursivo para mostrar essa quantidade em valores decrescentes da sequência: ... 1/125 1/25 1/5 1.
- 05.) Incluir em um programa (Exemplo0615) um método para ler uma cadeia de caracteres e chamar procedimento recursivo para mostrar cada símbolo separadamente, um por linha.
- 06.) Incluir em um programa (Exemplo0616) uma função recursiva para calcular a soma dos primeiros valores ímpares positivos começando em 5.
Testar essa função para quantidades diferentes.
- 07.) Incluir em um programa (Exemplo0617) uma função recursiva para calcular a soma dos inversos ($1/x$) dos primeiros valores ímpares positivos começando em 5.
Testar essa função para quantidades diferentes.
- 08.) Incluir em um programa (Exemplo0618) uma função recursiva 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.
- 09.) Incluir em um programa (Exemplo0619) uma função recursiva para contar os dígitos com valores ímpares em uma cadeia de caracteres.
Testar essa função para cadeias de diferentes tamanhos.
- 10.) Incluir em um programa (Exemplo0620) uma função recursiva para calcular a quantidade de maiúsculas em uma cadeia de caracteres.
Testar essa função para cadeias de diferentes tamanhos.

Tarefas extras

E1.) Incluir em um programa (Exemplo06E1) uma função recursiva para calcular o valor da função definida abaixo, lidos os valores de (x) e (n) do teclado:

$$f(x, n) = 1 + x^1 + x^3 + x^5 + x^7 + \dots$$

E2.) Incluir em um programa (Exemplo06E2) uma função recursiva para para calcular o valor indicado abaixo, lido o número de termos (n) do teclado:

$$e = 1 + 1/2! + 2/3! + 4/5! + 6/7! + \dots$$