

Núcleo Básico das Engenharias

C202-E/H

Algoritmos e Estruturas de Dados I

Capítulo 6 – Funções

Prof. Edson J. C. Gimenez

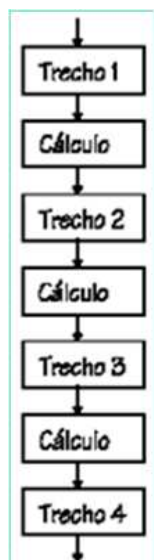
2019/Sem1

Material adaptado de: *Algoritmos e Estruturas de Dados*
- Profa. Rosanna Mara Rocha Silveira
- Prof. Evandro Luís Brandão Gomes

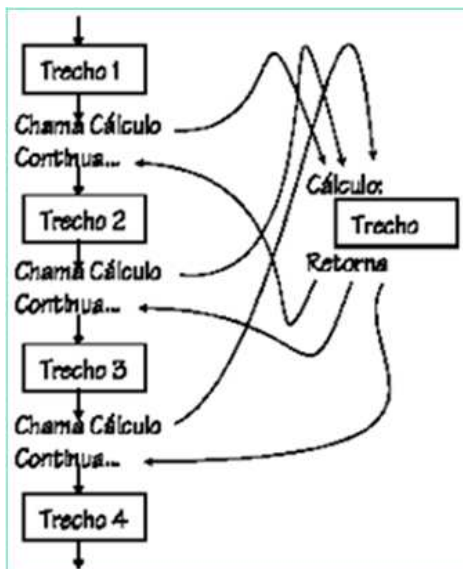
SUB-ROTINAS (SUB-ALGORITMOS / FUNÇÕES)

São blocos de instruções que realizam tarefas específicas.

O código de uma sub-rotina é carregado uma vez e pode ser executado quantas vezes for necessário.



(a) sem sub-rotina



(b) com sub-rotina

SUB-ROTINAS (SUB-ALGORITMOS / FUNÇÕES)

Porque usar sub-rotinas?

Como os problemas complicados exigem códigos extensos, a solução é **dividir os códigos grandes em trechos de códigos menores e de solução mais simples.**

Vantagens em se utilizar sub-rotinas:

- Subdivisão de códigos complexos com o objetivo de um melhor entendimento;
- Estruturação de códigos, com o objetivo de detecção de erros e melhores documentação e manutenção;
- **Reutilização do código.**

DEFINIÇÃO DE UMA SUB-ROTINA (FUNÇÃO)

Uma sub-rotina pode receber dados do programa que a chamou e ao terminar sua tarefa, ela pode fornecer ao programa que a chamou os resultados obtidos por ela.

CABEÇALHO	tipo – identifica o tipo de retorno da função (int, float, double, etc.)
	nome – nome da função, que é utilizado na sua chamada.
	parâmetros – variáveis que receberão os dados passados para a função.

variáveis locais	variáveis definidas dentro da função; validadas somente localmente.
-------------------------	---

CORPO	instruções que são executadas cada vez que a função é chamada
--------------	---

SUB-ROTINAS (SUB-ALGORITMOS)

SINTAXES DE SUB-ALGORITMO

Em geral, as linguagens de programação, possibilitam a modularização por meio de procedimentos e funções.

A linguagem C/C++ possibilita esta modularização por meio de funções.

FUNÇÃO

Tem o objetivo de desviar a execução do algoritmo principal para realizar uma tarefa específica e **quando ela retorna valor, ela retorna um e somente um valor ao algoritmo chamador.**

A função é chamada quando seu nome aparece no corpo do algoritmo chamador.

Pode-se utilizar **funções sem ou com passagem de parâmetros.**

Quando for com passagem de parâmetros, a função é chamada e lhe é atribuída alguns valores.

SUB-ROTINAS (SUB-ALGORITMOS)

SINTAXE DE FUNÇÃO

```
<tipo da função> <nome da função> (<lista_de_parâmetros>)  
{  
    <declaração das variáveis locais>  
    <bloco de comandos da função>  
    :  
    return dado;  
}
```

<tipo da função> - identifica o tipo de retorno da função (void, int, float, etc.).

<nome da função> - identifica o nome a ser utilizado para sua chamada; sua declaração segue as mesmas regras para declaração de variáveis.

<lista_de_parâmetros> - declaração das variáveis (parâmetros) que receberão os dados passados pela função que fez a chamada.

return - finaliza a execução da função, retornando o processamento à função que fez a chamada; se houver algum dado após o comando return, este valor é retornando à função que fez a chamada.

SINTAXES DE FUNÇÃO

Definição da função antes da função que faz a chamada:

```
<definição das funções>
....
....
....

int main ( )
{
    <declaração das variáveis >
    <corpo do função principal>
    <chamada das funções>
}
```

** Quando as funções “chamadas” são definidas antes das funções que as chamam não há necessidade da declaração do protótipos.

SINTAXES DE FUNÇÃO

Definição de funções depois da função que faz a chamada:

```
protótipos das funções;

int main ( )
{
    <declaração das variáveis >
    <corpo da função principal>
    <chamada das funções>
}

<definição das funções>
...
...
...
```

Protótipo: é um modelo que identifica como a função deve ser utilizada, devendo conter o **tipo** da função, seu **nome**, e seus **parâmetros** (se existirem).

** Quando as funções “chamadas” são definidas depois das funções que as chamam faz-se necessário a declaração do protótipos das funções.

Exemplo 1:

Escreva um programa que leia um valor numérico qualquer e mostre uma mensagem informando se esse valor é positivo (≥ 0) ou negativo (< 0).

Obs.: a verificação se o valor é positivo ou negativo deve ser feita por uma função que, tendo recebido como parâmetro um número qualquer, a mesma retorne 1, se o número recebido for positivo, ou 0 se o número recebido for negativo.

SINTAXE 1 (definição da função chamada antes da função chamadora; não é necessário a definição de protótipo):

```
#include <iostream>
#include <locale>
using namespace std;
int verifica(double valor)
{
    if(valor >= 0)
        return 1;
    else return 0;
}

int main()
{
    setlocale(LC_ALL, "portuguese");
    double num;
    int x;
    cout<<"Digite um número qualquer: ";
    cin>>num;
    x = verifica(num);
    if(x == 0)
        cout<<endl<<num<<" é negativo...\n ";
    else cout<<endl<<num<<" é positivo...\n";

    return 0;
}
```

SINTAXE 2 (definição da função chamada depois da função chamadora; há necessidade da declaração do protótipo da função):

```
#include <iostream>
#include <locale>
using namespace std;
int verifica(double valor); //protótipo da função
int main()
{   setlocale(LC_ALL, "portuguese");
    double num;
    int x;
    cout<<"Digite um número qualquer: ";
    cin>>num;
    x = verifica(num);
    if(x == 0)
        cout<<endl<<num<<" é negativo...\n";
    else cout<<endl<<num<<" é positivo...\n";
    return 0;
}

int verifica(double valor)
{
    if (valor >= 0)
        return 1;
    else return 0;
}
```

Exemplo 2: Faça um programa que leia um valor numérico positivo e escreva a raiz quadrada deste número através de uma função chamada RAIZ.

Obs.:

- A função principal irá ler o número, e passar para a função RAIZ;
- A função RAIZ é quem irá imprimir esse número, retornando "vazio" para a função main.

SINTAXE 1 (definição da função antes da função que faz a chamada):

```
#include <iostream>
#include <locale>
#include <cmath>
using namespace std;
void RAIZ(double valor)
{
    if(valor >= 0)
        cout<<"\nRaiz quadrada de "<<valor<<" é: "<<sqrt(valor)<<endl;
    else cout<<"\nImpossível - raiz de número negativo...\n";
    return;
}

int main()
{
    setlocale(LC_ALL, "portuguese");
    double num;
    cout<<"Digite um número qualquer: ";
    cin>>num;
    RAIZ(num);

    return 0;
}
```

SINTAXE 2 (definição da função depois da função que faz a chamada; necessidade de declaração do protótipo da função RAIZ):

```
#include <iostream>
#include <locale>
#include <cmath>
using namespace std;
void RAIZ(double valor); //protótipo da função
int main()
{
    setlocale(LC_ALL, "portuguese");
    double num;
    cout<<"Digite um número qualquer: ";
    cin>>num;
    RAIZ(num);
    return 0;
}

void RAIZ(double valor)
{
    if(valor >= 0)
        cout<<"\nRaiz quadrada de "<<valor<<" é: "<<sqrt(valor)<<endl;
    else cout<<"\nImpossível - raiz de número negativo...\n";
    return;
}
```

SUB-ROTINAS (SUB-ALGORITMOS)

Revisão: Deve-se usar funções quando:

- existir tarefas (trechos de códigos) que se repetem no programa;
- para facilitar o entendimento (modularização do problema);
- permitir a reutilização de código;

4 formas de uso:

- com parâmetros de entrada e de saída;
- somente com parâmetros de entrada;
- somente com parâmetro de saída;
- sem parâmetros

Exemplo 3 - Faça um programa que leia as 3 notas de um aluno, calcule e mostre sua média ponderada com os pesos 2, 3 e 5.

Obs.:

- A entrada das três notas e a saída da média serão realizadas na função `main()`.
- O cálculo da média deverá ser realizado na função `MEDIA()`, que irá receber como parâmetros as três notas, e retornará o valor da média.

Solução 1 – com parâmetros de entrada e saída

```
#include <iostream>
using namespace std;
float media( float NP1, float NP2, float NP3); //protótipo da função
int main(void)
{
    float n1, n2, n3, m;
    cout<<"Digite as tres notas: ";
    cin>>n1>>n2>>n3;
    m = media(n1, n2, n3); //chamada da função com parâmetros
    cout<<"Media: "<< m;
    return 0;
}

float media( float NP1, float NP2, float NP3)
{
    float md;
    md = NP1*0.2+NP2*0.3+NP3*0.5;
    return md; //retorna md
}
```

Solução 2 – somente com parâmetros de entrada

```
#include <iostream>
using namespace std;
void media( float NP1, float NP2, float NP3); //protótipo da função
float m; // m como variável global
int main(void)
{
    float n1, n2, n3;
    cout<<"Digite as tres notas: ";
    cin >> n1 >> n2 >> n3;
    media(n1, n2, n3); //chamada da função com parâmetros
    cout<<"Media: "<< m;
    return 0;
}
void media( float NP1, float NP2, float NP3)
{
    m = NP1*0.2+NP2*0.3+NP3*0.5;
    return; //não retorna nenhum valor
}
```

Solução 3 – somente com parâmetro de saída

```
#include <iostream>
using namespace std;
float media( );           //protótipo da função
float n1, n2, n3;         // variáveis globais
int main(void)
{
    float m;
    cout<<"Digite as tres notas: ";
    cin >> n1 >> n2 >> n3;
    m = media( );         //chamada da função sem parâmetros
    cout<<"Media: "<< m;
    return 0;
}
float media( )            //sem parâmetros de entrada
{
    float md;
    md = n1*0.2+n2*0.3+n3*0.5;
    return md;            //retorna md
}
```

Solução 4 – sem parâmetros e sem retorno

```
#include <iostream>
using namespace std;
void media( );            //protótipo da função
float n1, n2, n3, md;     // variáveis globais
int main(void)
{
    cout<<"Digite as tres notas: ";
    cin >> n1 >> n2 >> n3;
    media( );             //chamada da função sem parâmetros
    cout<<"Media: "<< md;
    return 0;
}
void media( )
{
    md = n1*0.2+n2*0.3+n3*0.5;
    return;               //não retorna nenhum valor
}
```

Escopo de variáveis

- Por escopo de variável entende-se o bloco de código onde esta variável é válida.

Variáveis globais:

- Declaradas fora de qualquer função, sendo válidas para todas as funções definidas a partir do ponto de suas declarações.

Variáveis locais:

- Aquelas válidas somente na função em que foram definidas.
- Uma variável definida dentro de uma função não é acessível à outras funções, mesmo que estas variáveis possuam o mesmo nome.

MECANISMO DE PASSAGEM DE PARÂMETROS

Na chamada de uma função, dependendo do caso, pode-se, ou não, fazer uso de parâmetros para a passagem de valores à função chamada.

Esta chamada pode se dar segundo **2 mecanismos: passagem por valor ou passagem por referência.**

PASSAGEM POR VALOR (ou por CÓPIA)

O parâmetro é fornecido ao sub-algoritmo no ato da invocação e as modificações feitas no sub-algoritmo com esse parâmetro não afetam o seu conteúdo, pois trabalha-se com a sua cópia, ou seja, passagem de parâmetros por valor significa que, para a execução da função, será gerada cópia do valor de cada um dos parâmetros.

SINTAXE:

Na função chamada:

tipo FUN (tipo parâmetro);

Na função que faz a chamada:

FUN (variável);

PASSAGEM POR REFERÊNCIA

Passagem de parâmetros por referência significa que os parâmetros passados para uma função correspondem a endereços de memória ocupados por variáveis. Dessa maneira, toda vez que for necessário acessar um determinado valor, isso será feito por meio da referência ao seu endereço.

Em C++, a passagem de referência pode ser feita com uso do operador de referência ou com a utilização de ponteiros. Em C, a passagem de referência é feita com a utilização de ponteiros.

Em C++:

Na função chamada: tipo nome_função (tipo &parâmetro)

Na função que chama: nome_função (variavel)

Em C e C++:

Na função chamada: tipo nome_função (tipo *parâmetro)

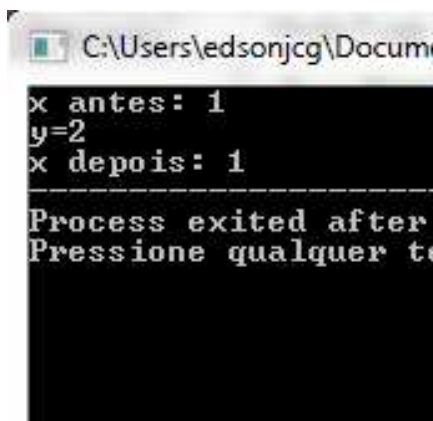
Na função que chama: nome_função (&parâmetro)

O símbolo ***** indica que o parâmetro é um ponteiro, que recebe um endereço de memória passada como parâmetro. Na chamada da função, o símbolo **&** indica que se está passando o endereço da variável, e não o seu valor.

Apost. - Exercício 6.3) Qual será o resultado da execução deste programa?

→ Passagem por valor:

```
#include <iostream>
using namespace std;
void passval( int y); //protótipo da função
int main(void)
{
    int x;
    x = 1;
    cout<<"x antes: "<< x <<endl;
    passval(x);
    cout<<"x depois: "<< x;
    return 0;
}
void passval( int y)
{
    y = y + 1;
    cout<< "y=" << y <<endl;
    return;
}
```



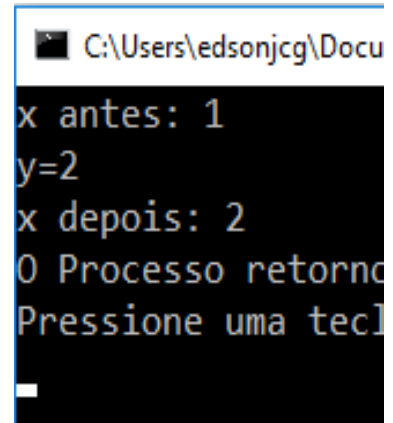
The screenshot shows a Windows command prompt window with the following output:

```
C:\Users\edsonjcg\Documents>
x antes: 1
y=2
x depois: 1
-----
Process exited after
Pressione qualquer tecla para continuar
```

Apost. - Exercício 6.4a) Qual será o resultado da execução deste programa?

→ Passagem por referência, com a utilização do operador de referência:

```
#include <iostream>
using namespace std;
void passref( int &y); //protótipo da função
int main(void)
{
    float x;
    x = 1;
    cout<< "x antes: " << x << endl;
    passref( x ); //chamada da função por referência
    cout<< "x depois: " << x;
    return 0;
}
void passref( int &y)
{
    y = y + 1;
    cout<< "y=" << y << endl;
    return;
}
```

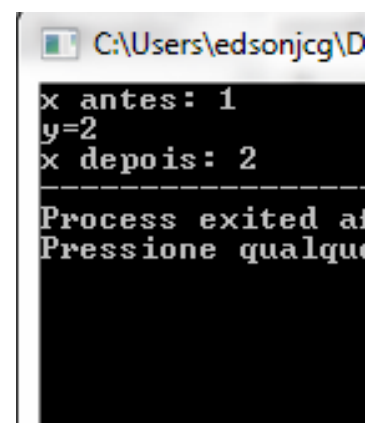


```
C:\Users\edsonjcg\Docu
x antes: 1
y=2
x depois: 2
0 Processo retornado
Pressione uma tecla
```

Apost. - Exercício 6.4b) Qual será o resultado da execução deste programa?

→ Passagem por referência, com a utilização de ponteiros:

```
#include <iostream>
using namespace std;
void passref( int *y); //protótipo da função
int main(void)
{
    float x;
    x = 1;
    cout<< "x antes: " << x << endl;
    passref( &x ); //chamada da função por referência
    cout<< "x depois: " << x;
    return 0;
}
void passref( int *y)
{
    *y = *y + 1;
    cout<< "y=" << *y << endl;
    return;
}
```



```
C:\Users\edsonjcg\D
x antes: 1
y=2
x depois: 2
-----
Process exited at
Pressione qualquer
```

```
#include <iostream>
using namespace std;
void circuito (int henry, int &farad);
int main( )
{
    int volt, ampere;
    volt = 8;
    ampere = 2;
    circuito(volt, ampere);
    cout<<"Volt = "<<volt<<" - Ampere = "<<ampere<<endl;
    return 0;
}
void circuito(int henry, int &farad)
{
    bool maxwell;
    henry = 3;
    farad = 7;
    maxwell = false;
    do{
        henry = henry -1;
        farad = farad - henry;
        if(henry == 0)
            maxwell = true;
    }while(!maxwell);
    return;
}
```

P6.13a) Execute o programa a seguir e indique os valores que serão impressos para as variáveis **VOLT** e **AMPERE** ao final da execução do programa e justifique estes valores.

```
C:\Users\edsonjcg\Documents\2018_9
Volt = 8 - Ampere = 4
0 Processo retornou 0 te
Pressione uma tecla para c
```

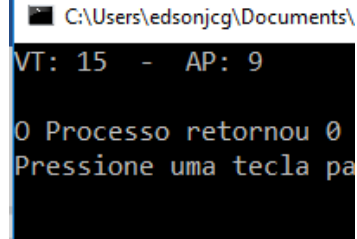
```
#include <iostream>
using namespace std;
void circuito (int henry, int *farad);
int main( )
{
    int volt, ampere;
    volt = 8;
    ampere = 2;
    circuito(volt, &ampere);
    cout<<"Volt = "<<volt<<" - Ampere = "<<ampere<<endl;
    return 0;
}
void circuito(int henry, int *farad)
{
    bool maxwell;
    henry = 3;
    *farad = 7;
    maxwell = false;
    do{
        henry = henry -1;
        *farad = *farad - henry;
        if(henry == 0)
            maxwell = true;
    }while(!maxwell);
    return;
}
```

P6.13b) Execute o programa a seguir e indique os valores que serão impressos para as variáveis **VOLT** e **AMPERE** ao final da execução do programa e justifique estes valores.

```
C:\Users\edsonjcg\Documents\2018_9
Volt = 8 - Ampere = 4
0 Processo retornou 0 te
Pressione uma tecla para c
```

```
#include <iostream>
using namespace std;
void tempo (int descd, int &ascd)
{
    bool existe;
    descd = 12;
    ascd = 15;
    while(!(existe))
    {
        descd = descd -5;
        if(descd < 0)
            existe = true;
        ascd = ascd - descd;
    }
    return;
}
int main( )
{
    int vt, ap;
    vt = 15;
    ap = 12;
    tempo( vt, ap);
    cout<<"VT: "<<vt<<" - AP: "<<ap<<endl;
    return 0;
}
```

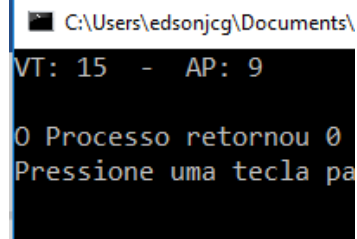
P6.14a) Execute o programa a seguir e indique os valores que serão impressos para as variáveis **VT** e **AP** ao final da execução do programa e justifique estes valores.



```
C:\Users\edsonjcg\Documents\
VT: 15 - AP: 9
0 Processo retornou 0
Pressione uma tecla pa
```

```
#include <iostream>
using namespace std;
void tempo (int descd, int *ascd)
{
    bool existe;
    descd = 12;
    *ascd = 15;
    while(!(existe))
    {
        descd = descd -5;
        if(descd < 0)
            existe = true;
        *ascd = *ascd - descd;
    }
    return;
}
int main( )
{
    int vt, ap;
    vt = 15;
    ap = 12;
    tempo( vt, &ap);
    cout<<"VT: "<<vt<<" - AP: "<<ap<<endl;
    return 0;
}
```

P6.14b) Execute o programa a seguir e indique os valores que serão impressos para as variáveis **VT** e **AP** ao final da execução do programa e justifique estes valores.



```
C:\Users\edsonjcg\Documents\
VT: 15 - AP: 9
0 Processo retornou 0
Pressione uma tecla pa
```

EP01: Escreva um programa que use uma função para trocar o valor de 2 variáveis.

*** Usando operador de referência.**

```
C:\Users\edsonjcg\Documents\2018_
Digite o valor 1: 10
Digite o valor 2: 20
trocando.....
Valor 1 = 20
Valor 2 = 10
0 Processo retornou 0 t
Pressione uma tecla para
```

```
#include <iostream>
using namespace std;
void troca (double &x, double &y);
int main( )
{
    double v1, v2;
    cout<<"\nDigite o valor 1: ";
    cin>> v1;
    cout<<"\nDigite o valor 2: ";
    cin>> v2;
    cout<<"\n trocando.....\n";
    troca( v1, v2);
    cout<<"\nValor 1 = "<< v1;
    cout<<"\nValor 2 = "<< v2;
    return 0;
}
void troca(double &x, double &y)
{
    double aux;
    aux = x;
    x = y;
    y = aux;
    return;
}
```

EP01: Escreva um programa que use uma função para trocar o valor de 2 variáveis.

**** Usando ponteiros.**

```
C:\Users\edsonjcg\Documents\2018_
Digite o valor 1: 10
Digite o valor 2: 20
trocando.....
Valor 1 = 20
Valor 2 = 10
0 Processo retornou 0 t
Pressione uma tecla para
```

```
#include <iostream>
using namespace std;
void troca (double *x, double *y);
int main( )
{
    double v1, v2;
    cout<<"\nDigite o valor 1: ";
    cin>> v1;
    cout<<"\nDigite o valor 2: ";
    cin>> v2;
    cout<<"\n trocando.....\n";
    troca( &v1, &v2);
    cout<<"\nValor 1 = "<< v1;
    cout<<"\nValor 2 = "<< v2;
    return 0;
}
void troca(double *x, double *y)
{
    double aux;
    aux = *x;
    *x = *y;
    *y = aux;
    return;
}
```


Exercícios propostos:

1) Fazer um programa que leia um par de números inteiros positivos, M e P, calcule e escreva o número de arranjos e o número de combinações desses M elementos, P a P, dados pelas fórmulas:

$$A = M! / (M - P)!$$

$$C = M! / (P! \times (M - P)!)$$

Por definição, se $M < P$, $A = 0$ e $C = 0$.

Obs.:

- A entrada dos valores M e P e a saída dos valores A e C serão realizadas na função principal.
- O cálculo do fatorial será realizado por uma função de nome FAT, que receberá como parâmetros um valor X inteiro e retornará o X! correspondente.
- O cálculo de A será feito por uma função de nome ARRANJO, que receberá os dois valores M e P e retornará o valor de A correspondente.
- O cálculo de C será feito por uma função de nome COMBINA, que receberá os dois valores M e P e retornará o valor de C correspondente.

2) Escreva um programa que leia 3 números, calcule e mostre, para esses 3 números, sua média aritmética simples e sua média aritmética ponderada, com os pesos 2, 3 e 5.

Obs.: A função principal só irá ler os 3 números e escrever as duas médias. Cada um dos 2 cálculos serão realizados por funções diferentes, chamadas pela função principal.

RECURSIVIDADE

Uma função é dita **recursiva** se é definida em seus próprios termos, isto é, quando existe dentro da função uma instrução de chamada para ela mesma.

Importante:

- Quando várias chamadas estão ativas, enquanto a última chamada não terminar, a penúltima não termina, e assim por diante.
- Isso faz com que as variáveis de cada chamada sejam todas mantidas na memória, o que requer mais memória.
- A função recursiva utiliza uma estrutura de dados chamada PILHA (abordagem *LIFO*).
- A função recursiva utiliza uma estrutura de dados chamada PILHA (abordagem *LIFO*).
- A recursividade produz repetição sem usar as estruturas repetitivas conhecidas (*while*, *do-while* e *for*).

Três pontos devem ser considerados na escrita de uma função recursiva:

1º) Definição do problema em termos recursivos: o problema é definido usando ele mesmo na solução.

2º) Definição da condição básica: condição básica nada mais é que a condição de término da função recursiva.

3º) Garantia de que, cada vez que a função é chamada recursivamente, ela deve estar mais próxima de satisfazer a condição básica, garantindo que o código não entre em loop.

Exemplo: Função não recursiva para cálculo do fatorial:

```
int fat (int n)
{
    int i, f;          // i é variável de controle e f é o fatorial
    f = 1;
    for (i = n ; i >= 2 ; i = i- 1)
        f = f * i;

    return f;
}
```

Exemplo: Função recursiva para cálculo do fatorial:

```
int fat (int n)
{
    int f;          // f é o fatorial

    if (n == 0)      } PARTE BASE
        return 1;

    else return n*fat(n-1); } PARTE RECURSIVA
}
```

Exemplo 4 – Cálculo do fatorial de N – sem recursividade:

```
#include <iostream>
using namespace std;
int fat (int n)
{
    int i, f = 1;
    for(i=2; i<=n; i++)
        f *= i;
    return f;
}
int main( )
{
    int nfat, i;
    float n;
    cout<<"\nDigite um valor para calculo do fatorial: ";
    cin>> n;
    if(n<0 || (n-int(n))!=0)
        cout<<"Não existe fatorial de "<<n<<endl;
    else
    {
        nfat = fat(n);    //chama a função fat
        cout<<n<<"! = "<<nfat;
    }
    return 0;
}
```

Exemplo 4 – Cálculo do fatorial de N – com recursividade:

```
#include <iostream>
using namespace std;
int fat (int n)
{
    if(n==0)
        return 1;
    else return n*fat(n-1);
}
int main( )
{
    int nfat, i;
    float n;
    cout<<"\nDigite um valor para calculo do fatorial: ";
    cin>> n;
    if(n<0 || (n-int(n))!=0)
        cout<<"Não existe fatorial de "<<n<<endl;
    else
    {
        nfat = fat(n);    //chama a função fat
        cout<<n<<"! = "<<nfat;
    }
    return 0;
}
```

Exemplo 5: Dados dois números inteiros, calcule o Máximo Divisor Comum entre eles através de uma função recursiva.

```
#include <iostream>
using namespace std;
int mdc (int p1, int p2);
int main( )
{
    int m, v1, v2;
    cout<<"\nDigite o valor 1: ";
    cin>> v1;
    cout<<"\nDigite o valor 2: ";
    cin>> v2;
    m = mdc( v1, v2);
    cout<<"mdc = "<<m;
    return 0;
}
int mdc (int p1, int p2)
{
    if (p2 == 0)
        return p1;
    else return mdc(p2, p1%p2);
}
```

Exercício proposto:

1) Fazer um programa que leia um par de números inteiros positivos, M e P, calcule e escreva o número de combinações desses M elementos, P a P, dado pela fórmula: $C = M! / (P! \times (M - P)!)$.

Obs.:

- A entrada dos valores M e P e a saída dos valores A e C serão realizadas na função principal.
- O cálculo de C será feito por uma função de nome COMBINA, que receberá os dois valores M e P e retornará o valor de C correspondente.
- O cálculo do fatorial será realizado por uma função recursiva de nome FAT, que receberá como parâmetros um valor X inteiro e retornará o X! correspondente.

Exercícios: