

TECNOLOGIAS DE PROGRAMAÇÃO DE SISTEMAS DE
INFORMAÇÃO

FUNÇÕES

PROGRAMAÇÃO ORIENTADA A OBJECTOS | Prof. Doutora Frederica Gonçalves

Cofinanciado por:



UNIDADE CURRICULAR : PROGRAMAÇÃO ORIENTADA A OBJECTOS

- Na linguagem C++, quando se invoca uma função que contém parâmetros de entrada, a **passagem de argumentos** pode ser feita de duas formas distintas:

valor

referência

UNIDADE CURRICULAR : PROGRAMAÇÃO ORIENTADA A OBJECTOS

Passagem por valor

Não é a variável ou a expressão que é enviada para a função, mas sim uma **cópia do seu valor**.

Os argumentos utilizados passam apenas os seus valores.

Nos exemplos seguintes, os argumentos utilizados passam a apenas os seus valores:

```
float calcula_area (int r)
```

```
cout<<calcula_area(2)
```

Nas chamadas à função, podemos utilizar no lugar do parâmetro um valor directo (2).

UNIDADE CURRICULAR : PROGRAMAÇÃO ORIENTADA A OBJECTOS

Passagem por valor

```
float calcula_area (int r)
```

```
cout<<calcula_area(x)
```

No lugar do parâmetro, podemos utilizar uma variável **x**.

Nota: No caso dos argumentos utilizados serem variáveis, não são as próprias variáveis que passam para dentro da função mas sim os seus valores.

UNIDADE CURRICULAR : PROGRAMAÇÃO ORIENTADA A OBJECTOS

- **Exemplo:**

Chamada a função
troca dentro da
função **main**

Função troca

```
#include <iostream>
using namespace std;
void troca (int x, int y); //protótipo
main () {
    int n1=1, n2 = 3;
    troca (n1, n2);
    cout << "n1 = " <<n1<< "\n";
    cout << "n2 = " <<n2<< "\n";
    System ("PAUSE");
}
void troca (int x, int y) {
    int aux ;
    aux = x; x = y; y = aux;
    cout << "x = " << x << "\n";
    cout << "y = " << y << "\n";
}
```

UNIDADE CURRICULAR : PROGRAMAÇÃO ORIENTADA A OBJECTOS

- Chamada a função **troca ()** na função **main()**:

```
troca (n1, n2); //com n1=1 e n1=3
```



```
void troca (int x, int y)
{
    int aux ;
    aux = x; x = y; y = aux;
    cout << "x = " << x << "\n"; // x = 3
    cout << "y =" << y << "\n"; // y = 1
    System ("PAUSE");
}
```

De volta a função
main ()



UNIDADE CURRICULAR : PROGRAMAÇÃO ORIENTADA A OBJECTOS

- O exemplo apresentado efectua a troca de dois valores inteiros:
 - Função **troca ()**, com dois parâmetros (**x** e **y**);
 - Os valores dos dois parâmetros (x e y), são trocados um com o outro dentro da função troca.
 - **Output:**

```
x = 3
y = 1
n1 = 1
n2 = 3
Prima qualquer tecla para continuar . . . _
```

UNIDADE CURRICULAR : PROGRAMAÇÃO ORIENTADA A OBJECTOS

- Em suma:

```
int n1=1 , n2 = 3;  
troca (n1, n2);
```

- É realizada uma chamada a função **troca ()**, passam apenas os valores dos argumentos **n1** e **n2**;
- São criados os parâmetros **x** e **y**, **x** recebe o valor de **n1** e **y** o valor de **3**, os valores de **x** e de **y** são trocados, durante a execução;
- Quando termina a execução do procedimento, todo o ambiente associado a função **troca ()** será eliminado.

UNIDADE CURRICULAR : PROGRAMAÇÃO ORIENTADA A OBJECTOS

Passagem por referência

- O que é enviado para a função não é uma cópia do valor da variável, mas sim a própria variável ou uma referência a esta.
- Qualquer alteração nos parâmetros da função, na realidade, corresponde a uma alteração nas próprias variáveis referenciadas.

UNIDADE CURRICULAR : PROGRAMAÇÃO ORIENTADA A OBJECTOS

- Em C ++, a passagem de argumentos por referência pode ser feita através de:

Ponteiros e endereços

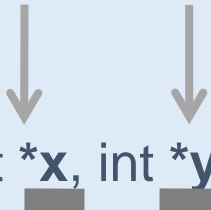
Parâmetros de referência

UNIDADE CURRICULAR : PROGRAMAÇÃO ORIENTADA A OBJECTOS

- Passagens de argumentos:

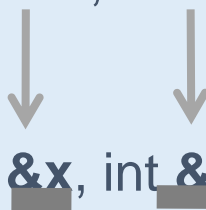
Ponteiros e endereços

```
#include <iostream>
...
void main () {
    int n1 = 1; int n2 = 2;
    troca ( &n1 , &n2 );
...
}
void troca (int *x, int *y) {
    int aux;
}
```



Parâmetros de referência

```
#include <iostream>
...
void main () {
    int n1 = 1; int n2 = 2;
    troca ( n1 , n2 );
...
}
void troca (int &x, int &y) {
    int aux;
}
```



UNIDADE CURRICULAR : PROGRAMAÇÃO ORIENTADA A OBJECTOS

- **Ponteiros e endereços:**

Ponteiros ←

Fornece como
argumentos os
endereços →

```
#include <iostream>
using namespace std;
void troca (int* x, int* y); //protótipo
main () {
    int n1 = 1 , n2 =2;
    troca (&n1, &n2);
    cout << "n1=" << n1 << '\n'; //n1=2
    cout << "n2=" << n2 << '\n'; //n2=1
    system ("PAUSE");
}

void troca (int *x, int *y) {
    int aux ;
    aux = *x; *x = *y ; *y = aux ;
    cout << "*x= " << *x << '\n'; //x=2
    cout << "*y= " << *y << '\n'; //y=1
}
```

UNIDADE CURRICULAR : PROGRAMAÇÃO ORIENTADA A OBJECTOS

- **Ponteiros e endereços**

- Os parâmetros da função **troca()** são definidos como **ponteiros**.

```
void troca (int *x, int *y);
```

- Quando escrevemos uma chamada à função temos de fornecer os **endereços das variáveis** que pretendemos passar para a função.

```
troca (&n1, &n2);
```

UNIDADE CURRICULAR : PROGRAMAÇÃO ORIENTADA A OBJECTOS

- **Ponteiros e endereços (continuação):**

- Esta chamada faz com que os parâmetros, que são ponteiros, da função **troca (int *x, int *y)** recebam os endereços das variáveis **n1** e **n2**.
- Dentro da função **troca (int *x, int *y)** todas as expressões que utilizam ponteiros (***x** e ***y**) referem-se aos valores passados como argumento, ou seja, os valores contidos em **n1** e **n2**.

UNIDADE CURRICULAR : PROGRAMAÇÃO ORIENTADA A OBJECTOS

- **Ponteiros por referência:**
 - **&** - é um **operador de referência** que serve para criar um outro nome para uma variável.
 - Este recurso dos operadores de referência pode ser utilizado na declaração de parâmetros.

```
Int n;  
Int &r = n;
```

A variável *r* é um outro nome para *n*

UNIDADE CURRICULAR : PROGRAMAÇÃO ORIENTADA A OBJECTOS

- **Parâmetros por referência:**

Na função main , quando é feita a chamada **troca (n1, n2)** , são as próprias variáveis n1 e n2 que passam para a função.

```
#include <iostream>
using namespace std;
void troca (int &x, int &y) ; //protótipo
main () {
    int n1 = 1 , n2 =4;
    troca (n1, n2);
    cout << "n1 = " << n1 << '\n'; //n1=4
    cout << "n2 = " << n2 << '\n'; //n2=1
    system ("PAUSE");}
void troca (int &x, int &y) {
    int aux ;
    aux = x; x = y ; y = aux ;
    cout << "x= " << x << '\n'; //x=4
    cout << "y= " << y << '\n'; //y=1
}
```


UNIDADE CURRICULAR : PROGRAMAÇÃO ORIENTADA A OBJECTOS

- **Parâmetros por referência**

- Na função **troca ()**, as instruções utilizam os **parâmetros x e y** sem necessidade de qualquer outra indicação;

```
void troca (int &x, int &y);
```

- **x e y** funcionam como *alias*es (outros nomes) das variáveis que forem passados à função como argumentos.

UNIDADE CURRICULAR : PROGRAMAÇÃO ORIENTADA A OBJECTOS

• Função

- Uma função que não seja declarada ***void*** deve retornar um valor.
- Por omissão do tipo de retorno, considera-se que a função retorna um valor tipo ***int***.
- O valor retornado é especificado pela instrução ***return***.
- A função pode também retornar (como seu valor de retorno) uma referência para uma variável global (***extern***).

UNIDADE CURRICULAR : PROGRAMAÇÃO ORIENTADA A OBJECTOS

- **Função que retorna uma referência:**

- **int &alterar();**

```
#include <iostream>
using namespace std;
int x = 10 ; // variável global
int &alterar() { return x ; } ;
main ()
{
    cout << "Valor inicial de x: ";
    cout << alterar() << '\n'; //escreve 10
    alterar() = 20;
    cout << "Valor alterado de x: ";
    cout << alterar() << '\n'; //escreve 20
    system ("PAUSE");
}
```

UNIDADE CURRICULAR : PROGRAMAÇÃO ORIENTADA A OBJECTOS

- **int &alterar();**
 - **&alterar()** - é escrita apenas a instrução: **return x;** ou seja, a função devolve uma referência para a variável **x**.
 - **&** - operador de referência faz com que a função funcione como um outro nome (*alias*) para a variável **x**.
 - **cout << alterar() << '\n';** - é escrito o valor de **x**.
 - **alterar()** - é possível colocar uma função do lado esquerdo do sinal de igual numa instrução de atribuição.
 - **alterar() = 20;** - é efectuada uma alteração no valor da variável **x**, que inicialmente tinha o valor 10 e passa a ter o valor 20.

UNIDADE CURRICULAR : PROGRAMAÇÃO ORIENTADA A OBJECTOS

- **Array**

- Não é mais do que um conjunto de elementos consecutivos em que são todos:
 - Do mesmo tipo;
 - Armazenados em memória;
 - E são acedidos através de um nome e de um índice.

```
char nome [2] // com 2 elementos
```

```
int numero [10] // com 10 elementos
```

UNIDADE CURRICULAR : PROGRAMAÇÃO ORIENTADA A OBJECTOS

- **Array**
- Na linguagem C++, é possível passar para uma função **arrays** como argumentos.
- Quando se passa um **array** não se cria uma cópia:
 - É passado, na realidade, o endereço do primeiro elemento do **array**.
 - A consequência dessa forma de passagem é que os elementos do **array** que forem modificados na função mantêm essas modificações, mesmo depois da função terminar.

UNIDADE CURRICULAR : PROGRAMAÇÃO ORIENTADA A OBJECTOS

- Função **med()**:
 - Permite calcular a média das notas

```
#include <iostream>
using namespace std;
float med (float n[ ],int t); //protótipo
main () {
    float notas [4] = {10,11,12,13};
    cout << "Media= " << med (notas, 4) << "\n";
}
float med (float n[ ], int t)
{
    float soma = 0;
    for (int i=0; i < t; i++)
        soma = soma + n[i];
    return (soma / t);
}
```


UNIDADE CURRICULAR : PROGRAMAÇÃO ORIENTADA A OBJECTOS

- Função **med()**
 - **med()** - inclui como parâmetros, um **array n[]** do tipo **float** e um **inteiro t** que indica o número de elementos do **array**.
 - **main()** - é declarado um **array** notas de **4** elementos, sendo inicializado com valores.
 - É feita uma chamada à função **med()** com a instrução: **cout << "Media= " << med (notas, 4);**
 - **for** - que acumula na variável *soma todos os valores do array*.
 - **notas** - que passou para o lugar do parâmetro **n[]**, quando foi incluído como argumento na chamada à função.
 - **média** - é calculada dividindo a soma por t.
 - os elementos do **array notas** não sofreram alterações, apenas forneceram os valores para os cálculos.

UNIDADE CURRICULAR : PROGRAMAÇÃO ORIENTADA A OBJECTOS

- Função **altera()**:
 - Neste programa o **array notas** sofre alterações.

Ciclo que percorre todos os elementos do *array*, acrescenta 0.5 a cada um

```
#include <iostream>
using namespace std;
void altera (float n[ ], int t); //protótipo
main () {
    float notas[4] = {10, 11, 12, 13};
    for (int i=0; i<4; i++)
        cout << notas [i] << "\n";
    altera (notas, 4);
    for (int i=0; i<4; i++)
        cout << notas[i] << "\n"; //escreve
        //os novos valores
    }
    void altera (float n[ ], int t){
        for (int i=0; i<t; i++)
            n[i] = n[i] + 0.5;
    }
```

UNIDADE CURRICULAR : PROGRAMAÇÃO ORIENTADA A OBJECTOS

- Função **altera()**
 - **main()** - temos um primeiro ciclo for que escreve os valores iniciais do **array**.
 - De seguida, é feita uma chamada à função altera com: **altera (notas, 4);**.
 - É incluído o nome do **array** **notas** para que ele passe para a função.
 - **altera()** - temos um ciclo for que percorre todos os elementos do **array**, *acrescentando-lhe 0.5* com: **$n[i] = n[i] + 0.5;$**



CTeSP

CURSOS TÉCNICOS
SUPERIORES PROFISSIONAIS