UNIVERSIDADE da MADEIRA

CTeSP CURSOS TÉCNICOS SUPERIORES PROFISSIONAIS

TECNOLOGIAS DE PROGRAMAÇÃO DE SISTEMAS DE INFORMAÇÃO

FUNÇÕES

PROGRAMAÇÃO ORIENTADA A OBJECTOS| Prof. Doutora Frederica Gonçalves

Cofinanciado por:

MADEIRA 14-20

REGIÃO AUTÓNOMA DA MADEIRA

PORTUGAL 2020

UNIÃO EUROPEIA

Fundo Social Europeu

UNIDADE CURRICULAR : PROGRAMAÇÃO ORIENTADA A OBJECTOS

- As funções como unidades fundamentais da estrutura de um programa

  – A função main (), representa a função principal e indispensável de um programa seja em C ou C++, sendo a primeira a ser executada.

  – Na linguagem C++, as funções são unidades de código fundamentais com que se escrevem e estruturam programas.

UNIDADE CURRICULAR : PROGRAMAÇÃO ORIENTADA A OBJECTOS

- **As funções como unidades fundamentais da estrutura de um programa**

  - Em linguagens como o Pascal, Visual Basic, etc., as unidades de código fundamentais como que se estruturam os programas podem ser de dois tipos: procedimentos ou funções.

  - **Em C++,** apenas existem **funções**. Não existem diferenças entre procedimentos ou funções.

UNIDADE CURRICULAR : PROGRAMAÇÃO ORIENTADA A OBJECTOS

- **As funções como unidades fundamentais da estrutura de um programa**

  - Reconhece-se uma função em C++, apenas pelos **parênteses** que se seguem ao nome (identificador).

Exemplo de funções:

main() rand() gets() strcpy()

UNIDADE CURRICULAR : PROGRAMAÇÃO ORIENTADA A OBJECTOS

Exemplo: um programa simples que **calcula a área** de um **rectângulo** a partir da medida do seu comprimento e da sua largura, introduzidas por si.

```
#include <iostream>
using namespace std;
main ()
{
    int comprimento;
    int largura;
    int area;
    cout << " Digite o comprimento: ";
    cin >> comprimento;
    cout << " Digite a largura: ";
    cin >> largura;
    area = comprimento * largura;
    cout << " Area do rectangulo = " << area << '\n';
    system ("Pause");
    }
```

UNIDADE CURRICULAR : PROGRAMAÇÃO ORIENTADA A OBJECTOS

O mesmo programa pode ser escrito utilizando algumas instruções e dar-lhes a forma de funções.

```
#include <iostream>
using namespace std;

     int comprimento; int largura;
     int area;
// iniciar a função
 void obter_area ()
{
     cout << "Digite o comprimento: ";
     cin >> comprimento;
     cout << "Digite a largura: ";
     cin >> largura;
  }
```

```
int calcula_area (int a, int b)
 {
       return (a*b);
   }
main ()
 {
       obter_area ();
      cout << "Area do rectangulo = “;
cout << calcula_area (comprimento,largura)
cout << '\n';
}
```

UNIDADE CURRICULAR : PROGRAMAÇÃO ORIENTADA A OBJECTOS

- **Estrutura geral de uma função**
  - A estrutura genérica ou sintaxe de uma função pode ser assim representada:

```
Tipo nome_da_função ([parâmetros])
{

<declarações e instruções da função>

[return [expressão] ; ]

}
```

- **Estrutura geral de uma função**
  - **Tipo**, é o tipo de dados que a função irá devolver.

    A função quando não devolve nenhum valor, então deve ser declarada como **void**.

  - **Nome_da função**: é um identificador que segue as mesmas regras dos nomes das variáveis.

```
void obter_area ();
```

UNIDADE CURRICULAR : PROGRAMAÇÃO ORIENTADA A OBJECTOS

- **Estrutura geral de uma função**
  - Se a função não necessitar de funcionar com parâmetros, os parênteses escrevem-se na mesma, mas **vazios** ou com a palavra **void**.

```
int obter_area (void);
```

  - Se a função necessitar de funcionar com **parâmetros**, eles são indicados dentro de parênteses, como se tratasse de uma declaração de variáveis.

```
int calcula_area (int a, int b);
```

- **Estrutura geral de uma função**
  - **Parâmetro**, é como uma **variável local**, declarada dentro do cabeçalho de uma função (dentro dos seus parênteses). Pode ser usado dentro do corpo de instruções da função.

  - Quando uma função é declarada com parâmetros e for chamada numa instrução do programa deve ser indicado um **argumento** por cada parâmetro e do mesmo tipo desse parâmetro.

UNIDADE CURRICULAR : PROGRAMAÇÃO ORIENTADA A OBJECTOS

- **Estrutura geral de uma função**
  - **Argumento**, é um valor que tem de ser fornecido no lugar de um parâmetro quando uma função é chamada.

```
calcula_area (comprimento, largura);
```

```
calcula_area (3, 4);
```

  - **Return**, por norma esta instrução aparece no final da função. Após a sua execução, o programa retorna ao seu ponto de chamada.

UNIDADE CURRICULAR : PROGRAMAÇÃO ORIENTADA A OBJECTOS

- **Protótipos e definições de funções**
  - Em funções C++ é importante fazer a distinção entre:
    - **Declaração de uma função** – que é feita através da escrita de um **protótipo**;
    - **Definição de uma função** – onde é escrito o **código completo** da função.

Protótipo de uma função é o seu cabeçalho declarado antes de se desenvolver (definir) o corpo de instruções da função em causa.

```
# include <iostream>
using namespace std;

int comprimento; int largura; int area;

void obter_area ();
int calcula_area (int a, int b);

main () {
(…)
```

- **Protótipos e definições de funções**
  - Um **protótipo** de uma função fornece ao compilador e a quem lê o código informações muito importantes, tais como:
    - Nome da função;
    - Tipo de dados que a função devolve;
    - Eventuais parâmetros com que a função opera e, sendo caso disso, os seus tipos de dados.

- **Funções inline e macros ao estilo de funções**
  - A criação de um programa, em C++, segue os seguintes passos:
    - Compilação
      - Pré-processamento: ler o código fonte; inclusão de bibliotecas, etc.;
    - Criação de um ficheiro executável
    - Execução do programa

- **Funções inline e macros ao estilo de funções**
  - **Inline,** é uma função usualmente pequena.
  - A palavra-chave **inline**, faz com que o código da função seja inserido no local onde surgir uma chamada de função.
  - Para criar uma função **inline**, escreve-se a palavra chave **inline** antes do cabeçalho da função:

```
inline int maior (int x, int y)

{return (x>y ? x:y;}
```

UNIDADE CURRICULAR : PROGRAMAÇÃO ORIENTADA A OBJECTOS

```
#include <iostream>
using namespace std;
inline int maior (int x, int y)
{return (x>y) ? x : y ; }
main ()
{
int n1, n2;
cout << “Introduza dois inteiros:”;
cin >> n1; cin >> n2;
cout << “Maior = “ << maior (n1,n2) << ‘\n’;
}
```

- **Funções inline e macros ao estilo de funções**
  - Outra forma de criar código neste estilo consiste em utilizar a directiva **# define.**
  - **# define,** serve também para definir constantes simbólicas, usualmente nas primeiras linhas do programa.

```
# define PI 3.14
```

  - Assim definida esta directiva, permite usar PI em instruções ou expressões como, por exemplo, cout << PI * raio * raio.

- **Funções inline e macros ao estilo de funções**
  - Esta mesma directiva, **# define,** permite definir **macros**, que aceitam argumentos.

```
# define QD(x)  ((x) * (x))
```

  - Assim, a **macro QD**, permite calcular o quadrado do argumento x.
  - Tendo uma instrução como << cout QD(5), fará escrever 25.

UNIDADE CURRICULAR : PROGRAMAÇÃO ORIENTADA A OBJECTOS

```
#include <iostream>
using namespace std;
# define MAIOR (x,y) ((x) >(y))?x:y)
main ()
{
int n1, n2;
cout << “Introduza dois inteiros:”;
cin >> n1; cin >> n2;
cout << “Maior = “ << MAIOR (n1,n2) << ‘\n’;
}
```

- **Variáveis globais e variáveis locais**

  - **Variável global** (ou **extern**) – é declarada antes de qualquer programa; neste caso, é utilizável em qualquer parte do programa.

  - **Variável local** (ou **auto**) – é declarada dentro de uma função ou de um bloco de código; neste caso, é utilizável nessa parte do programa.

- Exemplo

```
(…)
float soma;
float med (int n);
void main () {
     int quantos;
(…)
    cout << med (i);
(…)
}
float med (int n) {
float m;
m = soma/n;
return m;
}
```

Soma – é uma variável global
Pode ser usada em todo o programa

quantos – é uma variável local
Só pode ser usada em main

m – é uma variável local
Só pode ser usada em med

- **Variáveis globais e variáveis locais**
  - Em C++, é definido o conceito de **escopo das variáveis**, ou seja, a visibilidade e a duração das variáveis.
    - A **visibilidade** de uma variável refere-se às partes do programa em que está acessível e pode ser utilizada;
    - A **duração** de uma variável tem a ver com a parcela do tempo em que a variável existe no decurso do programa.

UNIDADE CURRICULAR : PROGRAMAÇÃO ORIENTADA A OBJECTOS

```
#include <iostream>

using namespace std;

float soma; // variável global
float med (int n); // protótipo da função
main ( )
{
int quantos; // variável local
float valor; // variável local
cout << " Quantos valores? ";
cin >> quantos;
```

```
for (int i = 1; i <= quantos; i++) {
cout << "Introduza um valor: ";
cin >> valor;
soma += valor;
cout << "Valor de media =" << med(i)<< '/n';
}
cout << "Soma =" << soma << '/n';
//cout << "Valor de m =" << m <<'/n'; // errado
}
float med (int n)  {
float m; // variável local
m = soma / n;
return m;
}
```

UNIVERSIDADE da MADEIRA

CTeSP CURSOS TÉCNICOS SUPERIORES PROFISSIONAIS