CINEL

A Tecnologia e o Futuro num só Centro

ENTIDADE FORMADORA CERTIFICADA

0809- Programação em C/C++ - Fundamentos

Luís Reis

INSTITUTO DO EMPREGO E FORMAÇÃO PROFISSIONAL

ASSOCIAÇÃO PORTUGUESA DAS EMPRESAS DO SECTOR ELÉCTRICO E ELECTRÓNICO

Lisb@20[20]

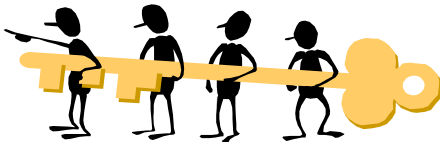
PORTUGAL 2020

UNIÃO EUROPEIA
Fundo Social Europeu

Conceitos Básicos

- Estrutura básica de um programa
- Variáveis e tipos de dados
- Comandos básicos
- Estruturas de controlo de fluxo

1º Programa em C++

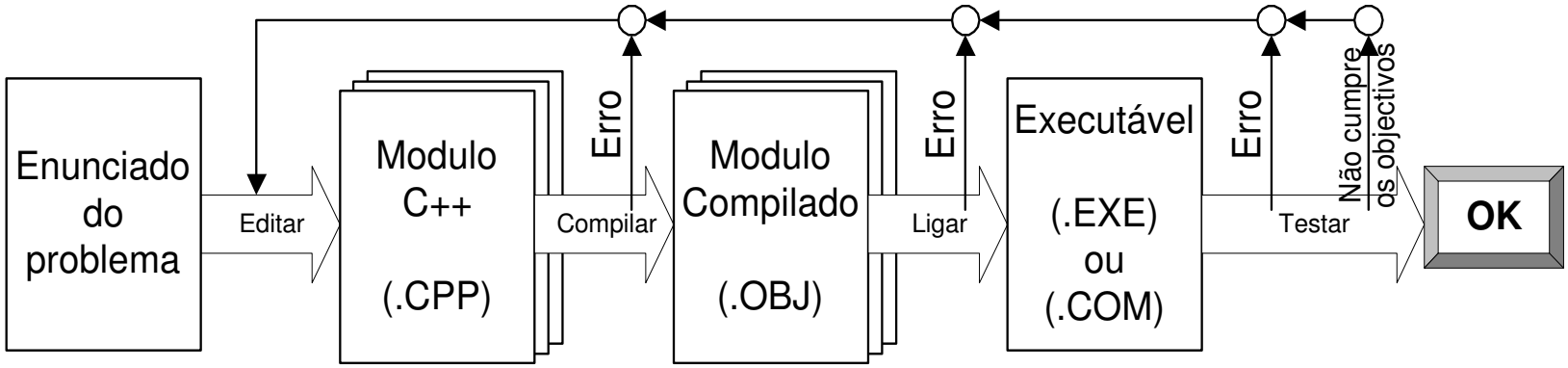


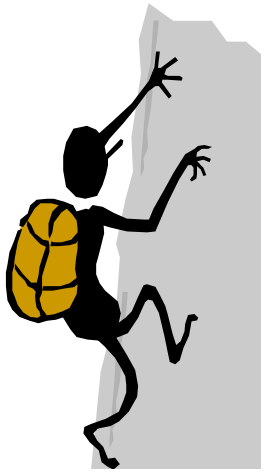
```
#include <iostream>
int main() {
  // Mensagem a mostrar no ecra.
  cout << "Hello, World!\n";
  return 0;
}
```

- Um programa em C++ é constituído por:
  - Várias funções, das quais uma obrigatoriamente tem que se chamar main.
    - A função main(), como qualquer outra é definida :
      - por um cabeçalho constituído por:
        - tipo de dados que a função devolve.
        - o seu nome.
        - parâmetros formais que recebe.
      - por um corpo (definido entre “{ }”) com :
        - declarações
        - definições
        - instruções
        - comentários
  - Inclusão de ficheiros *header* com:
    - protótipos de funções, macros, instruções, declarações de tipos

Ambiente de desenvolvimento

Fases de geração de um programa





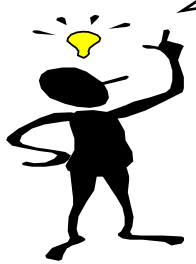
Sempre que um programa for constituído por vários módulos, torna-se necessário criar um projecto, indicando quais os módulos que devem ser ligados, na fase de criação do programa final.

Declarações e definições - variáveis

Em C/C++ qualquer identificador tem que ser declarado antes de ser usado.

Declarar uma entidade, consiste exclusivamente em **anunciar a sua existência**, explicitando-lhe o **nome** e o **tipo**.

No entanto, a maior parte das declarações, são também definições, isto é, definem a entidade que passa a ser designada por esse nome.

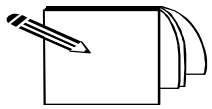


```
int y = 10;    // variável y do tipo inteiro
char c = 'k'; // variável c do tipo char, iniciada com o caracter 'k'
```

Tipos primitivos de dados e sua representação

Tipos fundamentais	Significado
int	inteiro
char	carácter
float	vírgula flutuante (*floating point*) de precisão simples
double	vírgula flutuante de precisão dupla

Qualificadores	Aplica-se a	Representa
short	int	menor dimensão
long	int ; double	maior dimensão
signed	char; int	com sinal
unsigned	char; int	valor sem sinal



A partir dos **tipos de variáveis básicos**, definem-se outros tipos de variáveis ditos **tipos derivados**, tais como ***arrays*, enumerados, apontadores, referências, estrutura e uniões** desses tipos fundamentais (ou básicos).

Alcance e espaço de memória ocupado

Tipo	**Ambiente 16** *bits*	**Ambiente 32** *bits*
signed int	2 *bytes* (de -32768 a +32767)	4 bytes (- 2 147 483 648 a + 2 147 483 647)
unsigned int	2 *bytes* (de 0 a 65535)	4 bytes (de 0 a 4 194 967 295)
float	4 *bytes* (de 3.4E-38 a 3.4E+38)	4 *bytes* (de 3.4E-38 a 3.4E+38)
signed char	1 *byte* (-128 a +127)	1 *byte* (-128 a +127)
unsigned char	1 *byte* (de 0 a 255)	1 *byte* (de 0 a 255)
short ou short int	2 *bytes* (Idêntico a int)	2 *bytes* (de -32768 a +32767)
unsigned short ou unsigned short int	2 *bytes* (Idêntico a unsigned int)	2 *bytes* (de 0 a 65535)
long ou long int	4 *bytes* (- 2 147 483 648 a + 2 147 483 647)	4 *bytes* (- 2 147 483 648 a + 2 147 483 647)
unsigned long int ou unsign long	4 *bytes* (0 a 4 194 967 295)	4 *bytes* (0 a 4 194 967 295)
double ou long double	8 *bytes* (1.7E-308 a 1.7E+308)	8 *bytes* (1.7E-308 a 1.7E+308)

Apresentação de números no ecrã *(cout)*

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

int main() {
   int int_num = 255;
   float float_num = 99.99895;
   // long int por omissão.
   long big_num = 1250500750.75;
   cout <<"123456789012345678901\n";
   cout << int_num << endl;
   cout << big_num << endl;
   cout << float_num << endl;
}
```

Qual será o *output* deste programa

```
12345678901234567890
255
1250500750
99.9989
```

Operadores e expressões

Como constituintes de um programa, constam também **operadores aritméticos, lógicos e relacionais (+, -, *, /, =).**

Com **variáveis** e **operadores**, constroem-se **expressões** a que ficam associados valores. Expressões operam sobre variáveis de forma a produzir novos valores.

Cálculo da área de um circulo

```
#include <iostream>   // Directiva para o pré processador.
Int main() {
  // Declaração (definição e iniciação) de uma constante do tipo float.
  const float  PI = 3.14159265;
  int raio;    // Declaração (definição) de um inteiro.
  float area; // Declaração (definição) de um float.

  // Instruções de saída e entrada de dados
  cout << "Qual o raio do circulo? ";
  cin >> raio;
  area = PI * raio * raio ; // Aqui têm que existir conversões de tipos.
  cout << "A area e" << area << endl; // Escrita do área no ecrã.
}
```

Operadores - Precedência e ordem de avaliação (1)

Símbolo	Descrição sumária	Forma de aplicação	Associatividade
:: ::	resolução de alcance refere nome global	nome_classe :: membro :: nome	→
-> [] () () sizeof sizeof	selecção de membro Indexação chamada a função construção de objecto dimensão de objecto dimensão de tipo	*apontador -> membro* *apontador [exp]* *exp (lista_exp)* *tipo (lista_exp)* *sizeof exp* *sizeof tipo*	→
++ -- ~ ! - + & * new delete delete[] ()	pós ou pré incremento pós ou pré decremento complemento *bit* a *bit* negação lógica unário menos unário mais endereço de desreferência criar , alojar destruir , desalojar destruir *array* *cast* , conversão de tipo	*lvalor ++ ou ++ lvalor* *lvalor -- ou -- lvalor* *~ exp* *! exp* *- exp* *+ exp* *& lvalor* ** exp* *new tipo* *delete apontador* *delete [ ] apontador* *(tipo) exp*	←
* / %	multiplicar dividir módulo, resto	*exp * exp* *exp / exp* *exp % exp*	→
+ -	adição , mais subtracção , menos	*exp + exp* *exp - exp*	→
<< >>	deslocar esquerda deslocar direita	*lvalor << exp* *lvalor >> exp*	→

Operadores - Precedência e ordem de avaliação (2)

Símbolo	Descrição sumária	Forma de aplicação	Associatividade
< <= > >=	menor que menor ou igual que maior que maior ou igual que	*exp* < *exp* *exp* <= *exp* *exp* > *exp* *exp* >= *exp*	→
== !=	Igual Diferente	*exp* == *exp* *exp* != *exp*	→
&	AND *bit* a *bit*	*exp* & *exp*	→
^	XOR *bit* a *bit*	*exp* ^ *exp*	→
|	OR *bit* a *bit*	*exp* | *exp*	→
&&	AND lógico	*exp* && *exp*	→
||	OR lógico	*exp* || *exp*	→
? :	Operador condicional	*exp* ? *exp* : *exp*	←
= *= /= %= += -= >>= <<= &= |= ^=	afectação simples multiplica e afecta divide e afecta módulo e afecta soma e afecta subtrai e afecta desloca direita e afecta desloca esquerda e afecta AND e afecta *bit* a *bit* OR e afecta *bit* a *bit* XOR e afecta *bit* a *bit*	*lvalor* = *exp* *lvalor* * = *exp* *lvalor* / = *exp* *lvalor* % = *ex* *lvalor* + = *exp* *lvalor* - = *exp* *lvalor* >> = *exp* *lvalor* <<= *exp* *lvalor* & = *exp* *lvalor* | = *exp* *lvalor* ^ = *exp*	←
,	vírgula, sequência	*exp* , *exp*	→

Atividade 1

- Crie um programa que receba dois números inteiros do teclado, guarde os mesmos em duas variáveis diferentes, somes os dois números e imprima o resultado no ecrã

  Output: "O resultado de N1 + N2 = resultado"

- Crie um programa que permita calcular a área de um quadrado (LXL);
- Crie um programa que receba do teclado um valor de temperatura em graus Celsius e converta para Fahrenheit, imprimindo o resultado da conversão no ecrã.
  - Fórmula de conversão - F=1,8C+32

Atividade 2

- Crie um programa que calcule o IMC baseado na seguinte fórmula:

  IMC = peso / (altura x altura)

  Nota: receber os dados do teclado

- Crie um programa que receba dois valores guardados nas variáveis “num1” e “num2”. Seguidamente troque os valores contidos nas variáveis num1→num2 e num2→num1.
  - Identifique o potencial problema;
  - Desenvolva uma estratégia para resolver esse problema.

Constantes

- Por definição são valores não alteráveis, ao contrário das “variáveis”;
- Existem constantes que podem ser declaradas pelo programador, podendo ser definidas através de uma diretiva `#define` ou através da palavra–chave `const`
  - Exemplo:

```
#include<iostream>
#define PI 3.14159265
int main()
{
      int raio = 3;
      double areaCirc;
      //PI poderia ter sido definido PI como
      //const float PI = 3.14159265
      areaCirc = raio*raio*PI ;
…
```

Funções de Entrada e Saída Formatada

Representação no ecrã (saída formatada) Função *printf()*

- Sintaxe: printf(<“%especificador”>, <variável>).
  - Ex: printf(“O número é %d”, num);
  - printf(“Olá Mundo!”);

especificador	**Saída**	**Exemplo**
d	Inteiro com sinal	392
u	Inteiro sem sinal	7235
x	Inteiro hexadecimal sem sinal	7fa
X	Inteiro hexadecimal sem sinal (maiúscula)	7FA
f	Ponto flutuante decimal (float, double)	392,65
c	Caracter	‘A’
s	Sequência de caracteres	“Nome”
p	Endereço do ponteiro	b8000000

Representação no ecrã (saída formatada) Função *printf()*

- Caracteres especiais
  - Existem caracteres especiais que não podem ser expressos sem ser no código fonte do programa, como por exemplo, nova linha (’\n’) ou tabulação (’\t’).
  - Todos são precedidos pela barra "\". A tabela 1.3 seguinte apresenta a lista dos caracteres especiais.
  - Usados dentro das “” da função, p.e :

```
printf(“\tOla \n”); // Produzirá uma tabulação antes e um
                    // quebra de linha depois do texto.
```

Caracter	Significado
\n	nova linha
\r	cursor para 1ª coluna
\t	tabulação
\b	*backspace*
\’	plica simples
\"	aspas simples

Entrada de dados (entrada formatada) Função *scanf()*

- Sintaxe: scanf(<especificador(s)>, variável(eis).

especificador	Saída	Exemplo
d	Inteiro decimal assinado	392
u	Inteiro decimal sem sinal	7235
x	Inteiro hexadecimal	7fa
f	Ponto flutuante decimal	392,65
c	Caracter	'A'
s	Sequência de caracteres - "string"	"Nome"
p	Endereço do ponteiro	b8000000

- Exemplo 1 (leitura de inteiros):

```
scanf("%d", &num); // Necessário "&" antes da variável.
```

- Exemplo 2 (leitura de cadeias de caracteres – "strings"):

```
char nome[20];
scanf("%s",nome); // Não é necessário "&" nas "strings"
printf("O nome é: %s", nome);
```