DEEC MEEC

DEPARTAMENTO DE ENGENHARIA ELECTROTÉCNICA E DE COMPUTADORES

Aulas e Provas de Avaliação 2009-2011

Programação de Computadores — Formulário

O meu primeiro programa em C++:

```
#include <iostream>   // para poder utilizar cout
using namespace std;  // para nao ter de escrever std::cout
int main() {
  cout << "Bom dia!" << endl;
  return(0);
}
```

Operadores aritméticos

Operador	Significado
*	multiplicação
/	divisão
+	adição
-	subtracção
%	resto da divisão inteira
++	incremento unitário
--	decremento unitário

Atribuição e operadores aritméticos

Operador	Significado
=	atribuição
*=	multiplicação e atribuição
/=	divisão e atribuição
+=	adição e atribuição
-=	subtracção e atribuição
%=	resto da divisão inteira e atribuição

Tabela 1: Operadores aritméticos e Atribuição.

Operadores relacionais

Operador	Significado
<=	menor ou igual que
<	menor que
>=	maior ou igual que
>	maior que
==	igual
!=	diferente

Operadores Lógicos

Operador	Significado	Utilização
||	OU lógico ($\vee$)	(exp1) || (exp2)
&&	E lógico ($\wedge$)	(exp1) && (exp2)
!	Negação ($\neg$)	! (exp1)

Tabela 2: Operadores relacionais e lógicos.

Operadores Lógicos Bit a Bit

Operador	Significado
&	E lógico bit a bit
|	OU lógico bit a bit
^	OU exclusivo bit a bit
~	Complemento bit a bit
<<	Rotação dos bits à esquerda
>>	Rotação dos bits à direita

Tabela 3: Operadores relacionais e lógicos bit a bit

Estes operadores são válidos sobre dados do tipo **int** ou **char**.

Dec.	Binário	Hex.	Dec.	Binário	Hex.
0	0000	0	8	1000	8
1	0001	1	9	1001	9
2	0010	2	10	1010	A
3	0011	3	11	1011	B
4	0100	4	12	1100	C
5	0101	5	13	1101	D
6	0110	6	14	1110	E
7	0111	7	15	1111	F

Tabela 4: Representação Decimal, Binária, e Hexadecimal

Tipo	tamanho em octetos (num PC de 32 bits)
bool	1
char	1
short int	2
int	4
long int	4
float	4
double	8
long double	12

Tabela 5: Tipos básicos.

\b	*backspace*	Mover o cursor uma vez para trás
\f	*form feed*	Ir para o início de uma nova página
\n	*new line*	Ir para a linha seguinte
\r	*return*	Ir para o início da linha corrente
\t	*tab*	Ir para a posição de tabulação seguinte
\'	*apóstrofe*	O símbolo '
\"	*aspas*	O símbolo "
\\	*barra à esquerda*	O símbolo \
\a	*bell (alarm)*	Emite um som (apito)
\nnn	*O carácter cujo código em octal é nnn*	Apresenta esse carácter
\xNN	*O carácter cujo código em hexadecimal é NN*	Apresenta esse carácter

Tabela 6: Caracteres especiais.

Forma geral de declaração e definição para uma variável de um tipo básico:

`<tipo>` *nome; // comentário*
`<tipo>` *nome = valor; // comentário*
`<tipo>` & *outronome = nome;* // variável do tipo referência
`<tipo>` * *nome_ptr;* // variável do tipo ponteiro
`<tipo>` * *nome_ptr* = `NULL;` // sem apontar
`<tipo>` * *nomeX_ptr* = `&` *nomeX;* // variável do tipo ponteiro

em que `<tipo>` deve ser substituído por um dos existentes na tabela 5, e `valor` deverá ser uma constante, ou uma expressão adequada à inicialização da variável em causa; o tipo de *nomeX* deverá igual ao tipo de **nomeX_ptr*.

Declaração e definição de uma estrutura (definição de um novo tipo):

```
struct <nome_estrutura> {
   <tipo> membro1;
   <tipo> membro2;
   . . .
   <tipo> membroN;
};
```

Para definir uma variável do tipo `struct nome_estrutura`:
`struct nome_estrutura` *nome _var*;
`nome_estrutura` *nome_var*;

Para definir uma variável do tipo ponteiro para estrutura:
`struct nome_estrutura *` *nome_var_ptr*;
`nome_estrutura *` *nome_var_ptr*;

Para definir e inicializar uma variável do tipo estrutura na definição:
`nome_estrutura` *nome_var* = { *valor1*, *valor2*,$\cdots$, *valorN*};

Para definir e inicializar uma variável do tipo ponteiro para estrutura na definição:
`nome_estrutura *` *nome_var_ptr* = `&` *nome_var*;

Para aceder ou atribuir valor a um dos membros utiliza-se a sintaxe:
nome_var.membro // no caso da estrutura
nome_var_ptr`->`*membro* // no caso de ponteiro para estrutura
(nome_var_ptr).membro* // membro de estrutura apontada

O tipo enum:
`enum` *nome_enum* {*etiqueta_1, etiqueta_2, ..., etiqueta_n*};
`enum` *nome_enum nome_var*;
`enum` *nome_enum* {*etiqueta_1, etiqueta_2, ..., etiqueta_n*} *nome_var*;

Tabelas

`<tipo>` *nomeVar*[*dimensao*];
`<tipo>` *nomeVar*[ ] ={$e_0, e_1, e_2, \cdots, e_n$};
`<tipo>` *nomeVar*[*dimensao*] ={$e_0, e_1, e_2, \cdots, e_n$};
em que $n \leq dimensao - 1$ e *dimensao* é um valor constante estritamente positivo.

Podemos definir tabelas de ponteiros:
`<tipo> *` *nomeVar*[*dimensao*];
que depois terão de ser devidamente inicializados.

Podemos definir tabelas de estruturas:
`struct nome_estrutura` *nomeVar*[*dimensao*];

Definição dinâmica de uma tabela:
`<tipo> *` *nomeVar*;
nomeVar `= new <tipo> [num];`
em que `num` é um valor inteiro estritamente positivo; se `num` for uma variável (do tipo `int`) tem de estar estar previamente definida e inicializada.

Para libertar o espaço pedido:
`delete[]` *nomeVar*;

Exemplos (tipos básicos):

```
bool ok = true;         // por omissão ok <-- true
char letra;             // letra que será lida do teclado
short int  i;           // contador de 1 até 100
short int & ref_i = i; // referência para a variável i
double x = 1.23456789; // para ter precisão
double * px = &x;       // ponteiro para x
```

Exemplos (tabelas e enum):

```
bool ok[5];              // haverá 5 estados (true/false) a vigiar
char vogal[] = {'a', 'e', 'i', 'o', 'u'};  // conjunto das 5 vogais
int  v[]={1,5,6};        // dimensões de um contentor
float temperatura[4];    // temperaturas às 8h, 12h, 16 e 20h
float temp_ano[365][4]; // temperaturas ao longo do ano (não bissexto)
double peso[10];         // pesos com 15 dígitos significativos

enum cores {VERMELHO, VERDE, AZUL}; // cores rgb
enum cores r = VERMELHO;
```

Strings no estilo C:

`char` *nomeVar*[*dimensao*];
`char` *nomeVar*[ ] = `"Texto ao gosto do utilizador";`
`char` *nomeVar*[*dimensao*] = `"Texto ao gosto do utilizador";`
em que o comprimento (útil) do texto deve ser menor que *dimensao*.

Exemplo de leitura duma string no estilo C:
`cin.getline(nomeVar, sizeof(nomeVar));`

Para saber o comprimento duma string no estilo C:
`strlen(nomeVar)`

Strings no estilo C++:

`string` *nomeVar*;
`string` *nomeVar* = "Texto ao gosto do utilizador";

Exemplo de leitura duma string no estilo C++:
`getline(cin, nomeVar);`

Para saber o comprimento duma string no estilo C++:
`nomeVar.length()`

if	`if` (*condição*) *Instrução;*	`if` (*condição*) { *Instrução1;* ... *InstruçãoN;* }
if else	`if` (*condição*) *InstruçãoT;* `else` *InstruçãoF;*	`if` (*condição*) { *InstruçãoT1;* ... *InstruçãoTn;* } `else` { *InstruçãoF1;* ... *InstruçãoFk;* }
if else if	`if` (*condição*) *InstruçãoT;* `else if` (*condição*) *InstruçãoFT;*	`if` (*condição*) { *InstruçãoT1;* ... *InstruçãoTn;* } `else if` (*condição*){ *InstruçãoFT1;* ... *InstruçãoFTk;* }

Tabela 7: Controlo de Fluxo (*if, if else, if else if*).

`switch` (*expressao*) {
`case <constante1> :` *Instrução;* `break;` // Onde está *Instrução*, podem estar
`case <constante2> :` *Instrução;* `break;` // zero ou mais instruções.
...
`case <constanteN> :` *Instrução;* `break;` // considerado necessário
`default:` *Instrução;* `break;`
}

`while` (*condição*) *Instrução;*	`while` (*condição*) { *Instrução1;* ... *InstruçãoN;* }
`do` *Instrução;* `while` (*condição*);	`do` { *Instrução1;* *Instrução2;* ... *InstruçãoN;* } `while` (*condição*);

Tabela 8: Controlo de fluxo (ciclos).

`for` (*instrução inicial*; *condição*; *instrução de iteração*) *Instrução;*
`for` (*instrução inicial*; *condição*; *instrução de iteração*){ *Instrução1;* *Instrução2;* ... *InstruçãoN;* }

Tabela 9: Mais Controlo de fluxo (ciclos).

Declaração (protótipo) de uma função:

`<tipo>` *nomefuncao*( ); // sem parâmetros
`<tipo>` *nomefuncao*(`<tipo1>` *nome1*, ..., `<tipoN>` *nomeN*); // com parâmetros
Se uma função não devolver qualquer valor o seu tipo de devolução deve ser declarado **void**.

Definição duma função:

`<tipo>` *nomefuncao*(){ // sem parâmetros
instrução 1 do corpo da função;
instrução 2 do corpo da função;
...
instrução N do corpo da função;
}

<tipo> *nomefuncao*(<tipo1> *nome1*, ..., <tipoN> *nomeN*) { // com parâmetros
instrução 1 do corpo da função;
instrução 2 do corpo da função;
...
instrução k do corpo da função;
}

A instrução **return** é utlizada para devolver um valor do tipo associado ao nome da função.

Manipulação de ficheiros – Leitura

Para para aceder a um ficheiro de entrada:
```
std::ifstream ficheiro_dados; // Fich. entrada
```

Em seguida é preciso associar esta variável com o nome do ficheiro que desejamos ler:
```
ficheiro_dados.open("nomefich.ext");// modo texto por omissão
ficheiro_dados.open("nomefich.ext", modo); // modo dado
```

Ou fazendo tudo de uma só vez:
```
std::ifstream ficheiro_dados("nomefich.ext");
std::ifstream ficheiro_dados("nomefich.ext", modo);
```

Manipulação de ficheiros – Escrita

Para para aceder a um ficheiro para escrita:
```
std::ofstream ficheiro_dados; // Fich. saida
```

Em seguida é preciso associar esta variável com o nome do ficheiro que desejamos escrever:
```
ficheiro_dados.open("nomefich.ext");// modo texto por omissão
ficheiro_dados.open("nomefich.ext", modo, proteccao); // modo dado
```

Ou fazendo tudo de uma só vez :
```
std::ofstream ficheiro_dados("nomefich.ext");// modo texto por omissão
std::ofstream ficheiro_dados("nomefich.ext", modo, proteccao);// modo dado
```
`proteccao` pode ter o valor 0666 (permissão de leitura e escrita para todos).

Bandeira (*flag*)	**Significado**
`std::ios::app`	Acrescenta informação no fim do ficheiro.
`std::ios::ate`	Abre e vai para o fim do ficheiro
`std::ios::in`	Abre o ficheiro para leitura
`std::ios::out`	Abre o ficheiro para escrita
`std::ios::binary`	Abre o ficheiro binário. Se esta bandeira não está activa o ficheiro é considerado como sendo ASCII.
`std::ios::trunc`	Descarta o conteúdo corrente do ficheiro quando aberto para escrita.

Tabela 11: Modo - *open flags*

Algumas funções membro de manipulação:

```
bool ficheiro_dados.close();       // fecha o ficheiro
bool ficheiro_dados.bad();         // true se o bit badbit foi activado
bool ficheiro_dados.fail();        // true se badbit ou failbit foram activados
bool ficheiro_dados.eof() const;   // true se foi activada a flag de fim de ficheiro
char int ficheiro_dados.peek();    // lê e devolve o carácter seguinte sem o extrair
```

Para saltar 1 caracter da entrada:
```
char ficheiro_dados.ignore();
```

Para saltar n caracteres da entrada ou até encontrar o carácter delim:
```
char ficheiro_dados.ignore(n, delim);
```

A leitura e escrita FORMATADA consegue-se usando os operadores >> e << , respectivamente.

A leitura NÃO FORMATADA:
fich_entrada.read(data_ptr, tamanho);
Em que:

- *fich_entrada* é um objecto da classe `ifstream` (ou `fstream`) aberto para leitura.
- *data_ptr*: ponteiro (para char) para o local onde deve ser colocada a informação lida.
- *tamanho*: número (int) de octetos a ler.

A escrita NÃO FORMATADA:
fich_saida.write(data_ptr, tamanho);

- *fich_saida* é um objecto da classe `ofstream` (ou `fstream`) aberto para escrita.
- *data_ptr*: ponteiro (para char) para o local onde está a a informação que será colocada no ficheiro
- *tamanho*: número (int) de octetos a escrever.

As funções membro `setf` e `unsetf` podem ser utilizadas para activar ou desactivar flags:
file_var`.setf(flags); // Activa flags`
file_var`.unsetf(flags); // Desactiva flags`
em que *file_var* é do tipo `ofstream` ou `ifstream`,

Bandeiras de conversão	
Flag	**Significado**
std::ios::skipws	Ignora os espaços que precedem a entrada.
std::ios::left	A saída é ajustada à esquerda.
std::ios::right	A saída é ajustada à direita.
std::ios::internal	Preenche uma saída numérica introduzindo um carácter de enchimento entre o sinal ou a base e o próprio número.
std::ios::boolalpha	Apresenta `true` e `false` em vez de 1 ou 0.
std::ios::dec	Apresenta números na base 10, formato decimal
std::ios::oct	Apresenta números na base 8, formato octal.
std::ios::hex	Apresenta números na base 16, formato hexadecimal.
std::ios::showbase	Apresenta o indicador de base no início de cada número.
std::ios::showpoint	Apresenta o ponto decimal em números de vírgula flutuante, quer seja ou não necessário.
std::ios::uppercase	Na conversão para hexadecimal apresenta os dígitos A-F em maiúsculas
std::ios::showpos	Coloca um sinal + antes de todos os números positivos.
std::ios::scientific	Converte todos os números de vírgula
std::ios::fixed	Converte todos o números para a notação de vírgula fixa.

Tabela 12: Bandeiras de conversão

Manipuladores de Entrada/Saída	
Manipulador	**Descrição**
std::setiosflags(long flags)	Escolhe as bandeiras a activar.
std::resetiosflags(long flags)	Desactiva as bandeiras seleccionadas.
std::dec	Apresenta números em formato decimal.
std::hex	Apresenta números em formato hexadecimal.
std::oct	Apresenta números em formato octal.
std::setbase(int base)	Escolhe a base de conversão: 8, 10 ou 16.
std::setw(int width)	Define a largura mínima da saída
std::setprecision(int precision)	Define a precisão de número em vírgula flutuante.
std::setfill(char ch)	Define o carácter de enchimento.
std::ws	Ignora os espaços que precedem a entrada.
std::endl	Apresenta o carácter de fim de linha `'\n'`.
std::ends	Apresenta o carácter de fim de string `'\0'`.
std::flush	Força a saída do que esteja em *buffer*.

Tabela 12: Manipuladores de Entrada/Saída

Mais funções de manipulação de ficheiros:

Seja:

```
ofstream fout("output.txt");
ifstream fin("input.txt");
```

- `tellp()` devolve a posição corrente do cursor de um ficheiro de **saída** (ponteiro para a posição do próximo `put`).
  `cout << fout.tellp();`
- `tellg()` O equivalente a `tellp()` para ficherios de entrada: devolve a posição corrente do cursor de um ficheiro de **entrada** (ponteiro para a posição do próximo `get`).
- `seekp(`*pos*`)` vai para a posição `pos` de um ficheiro de **saída**.
  `fout.seekp(3);` //// Vai para a posição 3
- `seek(`*pos*`)` vai para a posição `pos` de um ficheiro de **entrada**.
  `fin.seek(2);` // Vai para a posição 2
- `gcount()` é utilizada com canais de **entrada** devolve o número de caracteres lidos na última operação de entrada. `fin.seek(2);` // Vai para a posição 2
  É utilizada em conjunção com `get, getline` e `read`.
- `putback()` é utilizada com canais de entrada e coloca de novo na entrada o último carácter lido e coloca o cursor de leitura sobre ele. `cin.putback('g');`
  Está relacionada com a função `peek()`.

Directivas: #ifdef, #ifndef, ...	
#ifdef NOME # ··· ⋮ #endif	#ifndef NOME # ··· ⋮ #endif
#ifdef OUTRO # ··· ⋮ #else # ··· ⋮ #endif	#ifndef OUTRO # ··· ⋮ #else # ··· ⋮ #endif
#include <nomefichsistema>	#include "nomefich.h"
#define TAMANHO	#define TAMANHO 100

Tabela 10: **O Pré-processador**: algumas directivas.

`#undef` pode ser utilizada para remover uma definição anteriormente feita usando `#define`