Resumo/Revisão

Formação	**Jogo do Rato**
Local	`Marinha Grande`
Formador	`Sérgio Lopes, knitter.is@gmail.com [mailto:knitter.is@gmail.com]`
Ficha	**`2 - Coisas que Deviam Saber!`**

Variáveis

Em C existem os seguintes tipos de dados:

- Numéricos, a que correspondem todos os números inteiros ou reais, sejam positivos ou negativos.
- Cadeias de caracteres, a que chamamos *Strings* e que representam conjuntos de caracteres (números, letras, sinais de pontuação, etc) que podem ser usados e manipulados como texto.

A tabela apresentada abaixo mostra os tipos de dados numéricos que estão disponíveis para o desenvolvimento de programas em linguagem C. O tamanho exacto destes tipos de dados depende do processador em que o programa é compilado.

Tabela 1. Tabela de Tipos de Dados Numéricos

Tipo a Usar	Descrição	Número de bit	Consegue guardar
char	Carácter com sinal	8	-127 a 127
unsigned char	Carácter sem sinal	8	0 a 255
short	Inteiro pequeno	16	-128 a 127
unsigned short	Inteiro pequeno sem sinal	16	0 a 255
int	Inteiro	16 ou 32	-32768 a 32767
unsigned int	Inteiro sem sinal	16 ou 32	0 a 65535
long	Inteiro longo	32 ou 64	-2147483648 a 2147483647
unsigned long	Inteiro longo sem sinal	32 ou 64	0 a 4294967295
float	Valor real de precisão simples	*indefinido*	*indefinido*
double	Valor real de precisão dupla	*indefinido*	*indefinido*

Antes de usarmos uma variável é necessário declarar a mesma. A declaração de variáveis segue sempre o formato: *<tipo de dados> <nome da variável> [=<valor inicial>];*

Exemplos:

```
int x;
int z = 0;
double i = 50.5, g = 25.0, h;
```

```
char i = 'c';
```

Podemos também fazer uso de **constantes**, elementos similares a variáveis e que seguem as mesmas regras, mas cujo valor não pode ser alterado **depois** de ter sido definido a **primeira** vez.

É possível converter variáveis entre vários valores usando para isso a operação de *cast*. Este tipo de conversão pode provocar perda de informação e, em algumas situações, não é possível ser feita, mas regra geral é uma forma simples de converter valores numéricos entre os diferentes tipos.

Para usarmos o operador de **cast** devemos colocar o tipo de dados que pretendemos obter entre parêntesis seguido da variável ou valor a converter, **f = (int)x;**.

```
char x = 'D';
int a1;
float a2;
double a3;

//converter x para inteiro
a1 = (int) x;

//converter x para float
a2 = (float) x;

//converter x para double
a3 = (double) x;
```

Como é possível ver, para converter um tipo de dados para outro basta colocar o tipo de dados destino entre parêntesis seguido do valor a converter e o resultado será o valor convertido, com as perdas de informação caso tal aconteça.

Operadores Booleanos ou Lógicos

Os operadores lógicos permitem a comparação de expressões e devolvem sempre um de dois valores: verdadeiro ou falso.

Estes operadores podem ser organizados em: .. Operadores Relacionais: ==, >, <, >=, <=, **!=** - Comparam duas expressões ou variáveis e devolvem um valor verdadeiro ou falso de acordo com a comparação feita. .. Operadores Lógicos: **&&**, ||, **!** - Permitem aplicar as operações *boleanas* a dois valores lógicos (respectivamente: multiplicação lógica, soma lógica e negação).

Operador	**Significado**	**Exemplo**	**Descrição**
==	Igualdade	x == y	x é **igual** a y?
!=	Diferente	x != y	x é **diferente de** y?
>	Maior que	x > y	x é **maior que** y?
>=	Maior ou Igual que	x >= y	x é **maior ou igual** a y?
<	Menor que	x < y	x é **menor que** y?
<=	Menor ou Igual que	x <= y	x é **menor ou igual** a y?
&&	Multiplicação (E)	x && y	avalia x e y e **multiplica** os **seus valores lógicos**
||	Soma (OU)	x || Y	avalia x e y e **soma** os seus **valores lógicos**
!	Negação	!x	nega x

Decisões e Ciclos

if	Permite testar uma condição, sendo o código executado se a condição de teste for verdadeira. Pode ser completamentada com a componente **else** que permite ter código que é executado se a condição testada for falsa.
switch	Permite comparar um valor com várias opções expostas ao longo das várias cláusulas **case**. Pode conter uma última cláusula, **default**, que é usada quando o valor não corresponde a numa das opções presentes nas cláusulas anteriores. Sem a instrução **break**, as cláusulas **case** são executas sequêncialmente, após a primeira cláusula igual ao valor testado.
for	Ciclo usado quando, à priori, conhecemos quantas iterações serão executadas. Sintaxe na forma de *for(<iniciação>; <condição>; <pós-instruções>).*
while	Estrutura que permite a repetição de código mediante o teste de uma condição. O ciclo executa as instruções se a condição for verdadeira. Sintaxe na forma de *while(<condição>).*
do… while	Igual ao ciclo *while* mas como a condição é testada no fim das instruções, o código é executado sempre uma vez, seja a condição verdadeira ou não. Sintaxe na forma de *do { <instruções } while(<condição>);*.

Vectores

Permitem criar variáveis que conseguem agrupar um conjunto de valores, em cada uma das diferentes posições, desde que os valores sejam todos do mesmo tipo. Podem ter várias dimensões, sendo sendo mais comum os vectores de dimensão um e os de dimensão dois.

O tamanho e um vector, correspondente ao número de posições que o mesmo tem, não pode ser alterado depois de ser definido.

```
char a[] = {'e', 'x', 'e', 'm', 'p', 'l', 'o'},  s[50];

s[0] = 'e';
s[0] = 'x';
s[0] = 'e';
s[0] = 'm';
s[0] = 'p';
s[0] = 'l';
s[0] = 'o';
s[0] = '-';
s[0] = '2';
```

Strings

Em C, *strings* são apenas vectores de caracteres com um terminador especial, o **\0**. Sem esse terminar não podemos considerar que o vector seja uma *string* e não podemos usar correctamente as funções de manipulação de *strings* que as bibliotecas padrão nos oferecem.

Ponteiros

Permitem passar valores por referência para funções e gerir memória, bem como ser usados para tratamento de ficheiros e em estruturas de dados. Referenciam os valores de variáveis de forma indirecta.

```c
int x, *p; //criar uma variável e um ponteiro para inteiros

p = &x; //atribuir o endereço da variável ao ponteiro

(*p) = 5; //colocar, indirectamente, o valor 5 na variável
          //usando o ponteiro.
```

Todos os vectores são ponteiros, embora sejam ponteiros constantes que não podem ver alterados os valores para onde apontam.

Recursos a Consultar

- comp.lang.c Frequently Asked Questions, http://c-faq.com
- Richard L. Petersen. *Introductory C, Second Edition: Pointers, Functions, and Files*. Morgan Kaufmann. 7 de Novembro, 1996. ISBN 978-0125521420.