



**FACULDADE DE
CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA**

Departamento de Engenharia Electrotécnica

PROGRAMAÇÃO DE MICROPROCESSADORES

2021 / 2022

Mestrado Integrado em Engenharia Electrotécnica
e Computadores

1º ano

1º semestre

Trabalho nº 2 Tipos de dados básicos

1 Introdução

Os parágrafos sombreados contêm informações adicionais, que não são essenciais para o desenvolvimento dos trabalhos.

Nos capítulos 1 e 2 do livro “Linguagem C” de Luís Damas, recomendado para a disciplina de Programação de Microprocessadores, é feita uma primeira introdução à linguagem C, e são apresentados os tipos de dados básicos da linguagem. Esta aula visa consolidar estas matérias através de um conjunto de exercícios. Os primeiros três exercícios têm como objectivo complementar o estudo realizado na aula teórica. **No último exercício é dada menos ajuda. GUARDE O CÓDIGO desenvolvido em todos os exercícios na memória USB.** Durante esta aula e seguintes o docente pode pedir-lhe para mostrar o código desenvolvido para qualquer exercício, e pode fazer-lhe algumas perguntas.

O método de trabalho é o seguinte: nos primeiros exercícios os alunos usam o código fornecido no enunciado, fazendo as alterações pedidas; no último exercício o objectivo é resolver o problema proposto, aplicando os conceitos aprendidos nesta parte da matéria.

2 Ficheiros de Texto: WINDOWS vs LINUX vs MacOS

Os programas em C vão ser escritos em ficheiros de texto. Um ficheiro de texto é composto por caracteres em linhas. Ora o modo como uma linha é definida é diferente do Windows para o Linux, e para o Mac OS. Foi uma situação que foi decidida assim na altura e ficou assim...

Em baixo estão mostradas as diferenças:

Windows	Linux	Mac OS
Caracteres de uma linha CR LF	Caracteres de uma linha LF	Caracteres de uma linha CR
Caracteres de outra linha CR LF	Caracteres de outra linha LF	Caracteres de outra linha CR
Etc. CR LF	Etc. LF	Etc. CR

O **CR** (‘r’) designa-se por CARRIAGE RETURN e, como o nome indica, servia para levar o bloco de escrita de uma máquina de escrever para trás para o início da linha (lembra-se?).

O **LF** (‘n’) designa-se por LINE FEED e, como o nome indica, servia para passar para a linha seguinte (alimentar uma linha).

Ora os sistemas usavam os dois caracteres porque vinham da tradição das máquinas de escrever. O UNIX, simplificou muita coisa quando apareceu e uma delas foi retirar o ‘r’ do final das linhas, deixando ficar apenas o ‘n’.

Como devem perceber o mundo ficou um pouco confuso depois disto. Um ficheiro produzido num dos sistemas, colocado numa memória USB e passado para outro tem efeitos complicados, mas conhecidos. **Fica como se o ficheiro tivesse apenas uma linha** (na passagem num dos sentidos).

Esta diferença faz tanta dor de cabeça que o próprio editor WORD do OFFICE da MICROSOFT quando indicamos que queremos guardar um ficheiro em texto simples (com a terminação .txt) temos de indicar numa caixa de diálogo como queremos que as linhas terminem.

O **crimson** é um editor para Windows. Assim, produz ficheiros que têm o CR e o LF no final das linhas. Quer isto dizer que um ficheiro criado com o **crimson** pode ser aberto normalmente com os editores do Windows (WORD do OFFICE, o WordPad, ou o NotePad).

No mundo Linux existem dois programas para retirar (ou colocar) o CR das linhas. São especialmente úteis quando se está a usar o sistema Windows e o Linux alternadamente. Os seus nomes são dos2unix e unix2dos. Tente adivinhar qual é o que coloca e qual é o que retira. Aliás estes programas podem fazer mais do que simplesmente retirar ou colocar o CR. O modo de os usar é:

```
dos2unix fich1.txt fich2.txt
```

ou

```
unix2dos fich1.txt fich2.txt
```

Já sabe que para saber mais pormenores tem sempre o *man* para ler

3 Exercícios de aprendizagem

Esta secção inclui um conjunto de exercícios sobre os tipos *char*, *int*, *float*, e *double*. O objetivo é que o aluno aprenda a realizar operações sobre variáveis destes tipos.

Vai-se usar intensivamente a leitura de valores a partir do teclado e a escrita no ecrã, pois ainda não se sabe realmente programar. Acontece que operações de leitura e escrita de dados é igualmente um assunto algo delicado em C que necessita de muita prática.

A lista das funções de escrita e leitura mais usadas é a seguinte (repare que embora ainda exista, o *scanf* não deveria ser usado por motivos de segurança, sendo substituído pelo *fscanf*):

Leitura

getchar	vamos usar
scanf	vamos usar

Mas não devíamos usar **nunca**, pois o programa fica com uma porta de entrada para vírus.

gets	não vamos usar
------	----------------

Escrita

putchar	vamos usar
printf	vamos usar
puts	não vamos usar

Existe uma versão de cada uma destas seis funções com nome começado por ‘f’ (por exemplo, *fscanf*, ou *fprintf*), que são usadas para ficheiros, embora também possam ser usadas para o teclado e o ecrã. O uso deste grupo fica para mais tarde no semestre.

As duas primeiras funções (*getchar* e *putchar*) trabalham com caracteres. São muito simples de usar, mas quando os dados começam a ser inteiros, reais, etc., fica muito

aborrecido programar com elas pois tem de ser caracteres e transformá-los em inteiros ou reais. Não é difícil, mas dá trabalho e as outras funções já fazem isso. As outras são gerais e usam na sua lista de argumentos, entre aspas, uma sequência de caracteres com códigos e formatos para descrever o que realmente pretendemos fazer.

O sistema operativo Linux oferece ao utilizador um manual completo de programador, com todas as funções de biblioteca disponibilizadas pela biblioteca do C e os seus parâmetros. Este manual está organizado em pastas numeradas de 0 a 9: na pasta 0 estão os comandos da linha de utilizador; na pasta 3 está a maior parte das funções da biblioteca da linguagem C (o resto está nas pastas 2 e 0). Pode-se aceder a este manual utilizando o comando **man**, seguido do número de pasta, e do nome do comando.

EXERCÍCIO: Consulte o manual das funções *scanf*, *getchar*, e *printf*, utilizando respectivamente, as seguintes linhas de comando. Para sair do manual use a tecla ‘q’:

```
man 3 scanf
man 3 getchar
man 3 printf
```

3.1 O TIPO CARÁCTER

O tipo **char** permite armazenar 1 carácter, ocupando 8 bits, ou um *byte*. O formato de leitura e escrita de caracteres é “%c”. No entanto, caso se coloque um espaço antes do código na sequência de caracteres (e.g. “ %c”) o *scanf* salta espaços em branco, mudanças de linha e tabulações até chegar ao próximo carácter diferente destes.

EXERCÍCIO 1a: Crie um ficheiro novo com o nome **trocaChar.c** e introduza o código apresentado de seguida, omitindo os comentários (entre /* e */). Use a linha de comando para compilar o código, com o comando (“**gcc -o trocaChar trocaChar.c**”). Corra o programa (**./trocaChar** na linha de comando) introduzindo os três caracteres seguidos (e.g. “abc”) quando solicitado.

```
/*
 * Exercício 1 - troca de sequência de caracteres
 * Ficheiro: trocaChar.c
 */
#include <stdio.h> /* Usa biblioteca stdio */

main() { /* Início da função principal */
    /* Variáveis usadas para guardar 3 caracteres */
    char caracter1=' ', caracter2=' ', caracter3=' ';

    /* Lê caracteres */
    printf("Introduza três caracteres: ");
    scanf("%c", &caracter1);
    scanf("%c", &caracter2);
    scanf("%c", &caracter3);
    /* Escreve caracteres pela ordem inversa */
    printf("\nInvertendo a ordem dos caracteres obtemos: ");
    printf("%c' %c' %c'\n", caracter3, caracter2, caracter1);
}
```

EXERCÍCIO 1b: O programa anterior tem um problema. Se não se introduzir os três caracteres seguidos, o programa não escreve o que pretendemos. Tente corrê-lo com espaços entre os caracteres. CORRIJA este problema modificando o código fornecido.

EXERCÍCIO 1c: Imagine que na leitura dos caracteres queríamos fazer o seguinte: escrever o carácter e carregar na tecla “**enter**” do teclado depois de cada carácter. Tente correr o programa anterior e veja o que acontece.

EXERCÍCIO 1d: Use o mesmo programa **mas agora** com *getchar* em vez do *scanf*. Crie um ficheiro a partir do anterior chamado *trocaCharChar.c* em que trocou as instruções de leitura

```
scanf ("%c", &character1);
```

por

```
character1 = getchar ();
```

Tente pensar como se comportaria o programa nas situações de **1b** e **1c**. É mau demais para tentar fazer...

Use depois o *putchar* para escrever um carácter e experimentar o seu uso. Uma instrução possível é

```
putchar (character1);
```

Repare que escrever carácter a carácter é bem trabalhoso...

No Linux, cada vez que se carrega em “**enter**” é colocado o *line feed* (“\n”) e tem de se ler esse carácter com o *getchar* para continuar para o próximo carácter.

No Windows, cada vez que se carrega no “**enter**” são colocados **DOIS** caracteres: o *carriage return* (“\r”) e o *line feed* (“\n”).

O *scanf* “passa por cima” dos espaços em branco, tabulações e mudanças de linha (por isso é que o caso **1c** funcionou). Mas será que o *scanf* retira o último *line feed* a seguir a ter lido o último carácter que pretendia? Vamos ver que não, e isso pode trazer problemas.

3.2 TIPOS PARA NÚMEROS INTEIROS

O tipo *int* é o tipo base para guardar números inteiros. Na arquitectura das máquinas usadas no laboratório, cada variável do tipo inteiro ocupa 4 *bytes*. Em outras arquiteturas é possível que ocupe 2 *bytes*. O facto de não haver um número único pode ser um problema. É possível verificar quantos *bytes* são usados para um *int* com a função *sizeof*. Por outro lado para conseguir realizar código que funcione sempre da mesma maneira em qualquer computador, foram definidos modificadores ao tipo *int*, cada um especificando inteiros de tamanhos diferentes, com formatos de leitura e escrita distintos. A linguagem C também permite tratar os caracteres como inteiros apenas com um byte. A tabela seguinte exemplifica todos os tipos de inteiros suportados:

Tipo	Tamanho	Gama de valores	Formato leitura/escrita
<i>long long int</i>	8 bytes	$[-2^{63}, 2^{63}[$ ($2^{63}=9223372036854775808$)	“%lld”
<i>long int</i>	4 bytes	$[-2^{31}, 2^{31}[$ [= [-2147483648, 2147483647]	“%ld”
<i>short int</i>	2 bytes	$[-2^{15}, 2^{15}[$ [= [-32768, 32767]	“%hd”
<i>char</i>	1 byte	$[-2^7, 2^7[$ [= [-128, 127]	“%hhd”

Todos os tipos indicados anteriormente usam o *bit* mais significativo do número como *bit* de sinal. A linguagem C define ainda um modificador que pode preceder todos os tipos indicados anteriormente chamado de “*unsigned*”. O significado é que só pode ter valores não negativos, o bit mais significativo já é interpretado como valor e ganha-se precisão. O formato de leitura e escrita passa a ter *u* em vez de *d*. Por exemplo, o “*unsigned short int*” suporta uma gama de valores de $[0, 2^{16}[$, igual a $[0, 65535]$, e tem um formato de leitura e escrita igual a “*%hu*”.

Os inteiros têm as quatro operações básicas (+, -, *, /), mais uma que devolve o resto da divisão inteira (%). O valor resultante de uma operação com inteiros tem uma representação interna com o número de *bytes* correspondente à variável com mais *bytes* usada na operação. É possível modificar o número de *bytes* com um *cast*. Por exemplo “(*long int*) 3” tem 8 bytes, mas “(*short int*) 3” tem apenas 2 bytes.

EXERCÍCIO 2a: Crie um ficheiro novo com o nome *mediaSInt.c* e introduza o código apresentado de seguida, omitindo os comentários (entre /* e */). Repare que agora a leitura dos números é realizada com uma única instrução. Use a linha de comando para compilar o código, com o comando (“*gcc -o mediaSInt mediaSInt.c*”). Corra o programa (*./mediaSInt* na linha de comando) introduzindo os três números quando solicitado.

```
/*
 * Exercício 2 - média de inteiros short
 * Ficheiro: mediaSInt.c
 */
#include <stdio.h>

main() {
    const int NUMERO= 3; /* CONSTANTE com número de elementos */
    /* Variáveis para guardar Dados */
    short int val1=0, val2=0, val3=0;
    short int total; /* Variável para guardar valor total */
    short int media; /* Variável para guardar média */

    /* Lê valores */
    printf("introduza os três valores:");
    scanf(" %hd %hd %hd", &val1, &val2, &val3);

    total= val1+val2+val3; /* Calcula a soma */
    media= total/NUMERO; /* Calcula a divisão inteira */

    printf("\n"); /* Acrescenta uma linha em branco */
    /* Escreve o resultado */
    printf("A soma dos valores é %hd e a média é %hd\n",
           total, media);
}
```

Corra o programa e verifique o seu comportamento com o tipo de entradas experimentadas na secção anterior. Isto é, escreva um inteiro seguido de “enter” seguido de outro inteiro, seguido de “enter”, etc. O que acontece?

Agora introduza uma letra em vez de um número. O que acontece?

Pense que utilizaria o *getchar* em vez do *scanf*. É capaz de não ser muito útil... Certos números têm mais do que um carácter, por exemplo o 160 tem três. Mas, por outro lado, pode ser interessante ler carácter a carácter, verificar se é numérico e fazer as contas... Vai resolver um problema destes num trabalho mais avançado, mas não por agora.

EXERCÍCIO 2b: O programa anterior tem outros problemas. Caso a soma ou um dos números introduzidos seja superior a 32767, ou inferior a -32768, o programa não consegue calcular a média correta. Experimente fazer isso. Modifique o programa de forma a calcular a média de números positivos (sem sinal), com o valor máximo possível de $2^{32}-1=4294967296$. Chame ao ficheiro *mediaPInt.c*.

EXERCÍCIO 2c: Apresente o resultado final da média sobre a forma de valor inteiro, mais uma fracção. Por exemplo, o valor médio de “1+1+2” é “1 1/3”. Ou o valor médio de 34, 35 e 356 é 141 2/3.

Relembra-se que a operação divisão devolve o valor truncado às unidades, e é possível obter o resto da divisão inteira utilizando a operação módulo (“%”). Chame ao ficheiro *mediaFInt.c*.

3.3 TIPOS PARA NÚMEROS REAIS

Os tipos *float* e *double* são usados para guardar números reais, ocupando respectivamente 4 e 8 *bytes*. As gamas de valores para os dois tipos anteriores são respectivamente $[-3.2 \times 10^{\pm 98}, 3.2 \times 10^{\pm 98}]$ e $[-1.7 \times 10^{\pm 908}, 1.7 \times 10^{\pm 908}]$.

O formato de leitura de números reais é respectivamente “%f” e “%lf” para os tipos *float* e *double*. O *scanf* aceita a introdução de números no formato tradicional (e.g. 1234.5), ou em notação científica (1.2345e3 ou 1.2345E3). O formato de escrita para os dois tipos de números reais pode ser “%f”, “%e”, ou “%E” respectivamente escrevendo os números no formato tradicional, em notação científica com ‘e’, ou em notação científica com ‘E’.

Os reais têm as quatro operações básicas (+, -, *, /). O resultado é real sempre que pelo menos um dos operandos for real. Quando se misturam inteiros com reais, deve-se ter muito cuidado. Por exemplo, no exercício 2, as expressões “(val1+val2+val3)/3” e “(val1+val2+val3)/3.” (repare no ponto) têm valores diferentes: a primeira dá um valor inteiro, enquanto a segunda dá um valor real. É possível manipular os tipos usando o *cast* (e.g. “(long int)3.4”). Mas atenção que o *cast* não é um conversor. Tem de se saber usar.

EXERCÍCIO 2d: Modifique o programa que calcula a média, de forma a também apresentar a média sobre a forma de um número fraccionário, para além das representações indicadas anteriormente. Por exemplo, o valor médio de 34, 35 e 356 é 141,66666667 ou 141 2/3. Chame ao ficheiro *mediaJInt.c*.

EXERCÍCIO 3a: Copie o programa desenvolvido no exercício 2d para o ficheiro *mediaFloat.c*, e modifique-o de maneira a calcular a média de três valores reais. O programa deve ler três *floats*, e apresentar a soma e o valor da média tanto no formato tradicional, como no formato de notação científica. Experimente com os números 1e34, 1e34 e 2e34.

EXERCÍCIO 3b: Se tiver tempo, depois de terminar o exercício final, copie o ficheiro *mediaFloat.c* para *mediaDouble.c* e repita o exercício 3a, mas agora utilizando o tipo *double* para guardar os números reais. Poderá verificar que o erro cometido nos cálculos é inferior.

4 Sequência de controlo de formato

A sequência de controlo de formato no caso do *printf* é simples: escreve o que lá pusermos! No caso de leitura o problema complica-se um pouquinho. Se colocarmos certos caracteres na sequência o programa está à espera de os encontrar na entrada e se não os encontrar o programa funciona mal.

Por exemplo, a partir do ficheiro *mediaSInt.c* crie um novo ficheiro chamado *mediaSIntFor.c* e mude a instrução de *scanf* para a seguinte

```
scanf("um %hd dois %hd tres %hd", &val1, &val2, &val3);
```

e ao correr o programa introduza

```
um 12 dois 13 tres 14
```

Depois corra outra vez o programa e introduza, por exemplo

```
12 13 14
```

e veja o que acontece quando não se cumpre exactamente a sequência que se está à espera.

Como vê, pode-se passar de uma forma descontraída de fazer entrar os dados para outra forma muito rígida.

5 Exercício final

Pretende-se desenvolver um programa que calcule a diferença entre dois tempos, definidos na forma n.º de dias, n.º de horas, n.º de minutos e n.º de segundos. O programa deve pedir sucessivamente ao utilizador para introduzir cada um dos tempos, e depois, deve apresentar a diferença entre o segundo tempo e o primeiro tempo em segundos, e também no formato n.º de dias, n.º de horas, n.º de minutos e n.º de segundos. O programa deve permitir lidar com tempos até 1000 dias = 86400000 segundos.

Método de desenvolvimento do programa:

1. Comece por definir todas as variáveis que vai necessitar para guardar os tempos, e a diferença entre tempos. Defina os tipos apropriados, de forma a não perder resolução;
2. Leia os dois tempos para dois conjuntos independentes de variáveis, relativos a cada parcela. Lembre-se que na sequência do *scanf* podem aparecer letras, para além dos formatos de leitura apresentados nos exemplos anteriores;
3. Converta os dois tempos para segundos, e calcule a diferença;
4. Converta o tempo final em dias, horas, minutos e segundos;
5. Escreva o resultado final;
6. **Chame o docente e mostre a aplicação a funcionar durante a aula.**

Apresenta-se na figura seguinte um exemplo de utilização da aplicação pretendida:

```
./final1
Introduza o tempo no.1 (dia)d (hora)h (minuto)m (segundo)s:
0d 23h 55m 59s
Introduza o tempo no.2 (dia)d (hora)h (minuto)m (segundo)s:
2d 1h 1m 2s
A diferença entre os dois tempos é de 90303 segundos = 1d 1h 5m 3s
```