FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

Departamento de Engenharia Electrotécnica

PROGRAMAÇÃO DE MICROPROCESSADORES

2012 / 2013

Mestrado Integrado em Engenharia Electrotécnica
e Computadores

1º Ano
1º Semestre

Trabalho nº 1
Tipos de dados básicos

José Barata
Paulo da Fonseca Pinto
Luis Bernardo

1 EXERCÍCIO DE APRENDIZAGEM

Esta página deve ser entregue ao Docente antes do início da aula. Antes de responder às perguntas leia o capítulo do livro da cadeira correspondente à matéria do enunciado. Lembre-se que existe informação útil ao longo deste documento.

A. Qual a diferença entre ler um ENTER em Windows e Linux.

B. Como faria se tivesse uma variável do tipo char com o valor de ‘5’ e quisesse guardar este mesmo valor num int. Dê um exemplo.

2 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.

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.

3 Exercícios de aprendizagem

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

Como ainda não se sabe realmente programar, vamos usar intensivamente a **leitura de valores a partir do teclado e a escrita no ecrã**. Vamos fazer uso de operações de leitura e escrita de dados, o que é um assunto também algo delicado em C.

Para ficarmos com a lista completa das funções de escrita e leitura mais usadas, aqui vai com a indicação se vai ser usada neste trabalho ou não:

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	

Já agora, para se ficar com a ideia de todas as operações de escrita e leitura, existe depois uma versão de cada uma destas seis funções com nome começado por 'f' (por exemplo, ***fscanf***, ou ***fprintf***), mas isso fica para mais tarde.

Em programação a sério deve-se usar o ***fscanf*** em vez do ***scanf*** por causa da segurança.

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. 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.

Opcional: 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 entre caracteres até chegar ao próximo carácter.

EXERCÍCIO 1a: Crie um ficheiro novo com o nome ***swapChar.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 swapChar swapChar.c***"). Corra o programa (***./swapChar*** na linha de comando) introduzindo os três caracteres seguidos (e.g. "123") quando solicitado.

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

main() { /* Início da função principal */
    /* Variáveis usadas para guardar 3 caracteres */
    char var1=' ', var2=' ', var3=' ';

    /* Lê caracteres */
    printf("Introduza três caracteres: ");
    scanf("%c", & var1);
    scanf("%c", & var2);
    scanf("%c", & var3);
    /* Escreve caracteres pela ordem inversa */
    printf("\nCaracteres invertidos: ");
    printf("'%c' '%c' '%c'\n", var3, var2, var1);
}
```

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

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

```
scanf ("%c", &var1);
```

por

```
var1 = getchar ();
```

Repare como se comporta o programa nas situações anteriores.

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

```
putchar (var1);
```

Repare que escrever carácter a carácter é bem mais 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" desses caracteres até encontrar o que quer (por isso é que o caso **1c** funcionou. O que ele queria era o próximo carácter "*mais normal*"). Vamos ver no futuro que o ***scanf*** **NÃO** retira o último *line feed*, mas apenas os intermédios na procura que faz pelo que quer.

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* (noutras arquitecturas pode verificar com a função ***sizeof***). Para conseguir realizar código que funciona 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
long long int	8 bytes	$[-2^{63},2^{63}[$ *(2^{63}=9223372036854775808)*	*"%lld"*
long int	4 bytes	$[-2^{31},2^{31}[$ = [-2147483648, 2147483647]	*"%ld"*
short int	2 bytes	$[-2^{15}, 2^{15}[$ = [-32768 , 32767]	*"%hd"*
char	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 "***unsigned***" que pode preceder todos os tipos indicados anteriormente, para fazer com que o bit mais significativo seja interpretado como valor. O formato de leitura e escrita passa então a ter ***u*** em vez de ***d***. Quando é usado, deixa de ser possível representar números negativos. 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***".

A linguagem C define as quatro operações básicas (+, -, *, /), mais uma operação 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) 8*" tem 4 bytes, mas "*(short int) 8*" apenas tem 4 *bytes*.

EXERCÍCIO 2a: Crie um ficheiro novo com o nome ***averageSInt.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 averageSInt averageSInt.c***"). Corra o programa (***./averageSInt*** na linha de comando) introduzindo os três números quando solicitado.

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

main() {
    const int NUMBER= 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 average; /* 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 */
    average= total/NUMBER;      /* 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, average);
}
```

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…

EXERCÍCIO 2b: 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. Garanta também que o seu novo programa consegue calcular

a média de números positivos (sem sinal), com o valor máximo no total de 2^{32}-*1= 4294967296*.

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 ***averageFInt.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]-3.2x$10^{\pm 98}$, 3.2x$10^{\pm 98}$[e]-1.7x$10^{\pm 908}$, 1.7x$10^{\pm 908}$[.

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'.

A linguagem define as quatro operações básicas (+, -, *, /). Uma operação numérica tem um resultado real sempre que pelo menos um dos elementos da expressão é real. Quando se misturam inteiros com reais, parte da expressão pode ser calculada com inteiros e outra parte com reais. Por exemplo, nas alíneas anteriores, o resultado da expressão *"(val1+val2+val3)/**3**"* tem valores diferentes em cada uma das alíneas: a primeira dá um valor inteiro, enquanto a segunda dá um valor real. É possível converter um real para um inteiro usando um *cast* (e.g. "*(long int)3.4*"), truncando-se a parte fraccionária.

EXERCÍCIO 2c: 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 ***averageJInt.c***.

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 pouco. 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.

EXERCÍCIO 3: a partir do ficheiro ***averageSInt.c*** crie um novo ficheiro chamado ***averageSIntFor.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 29s
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
```