10ª Aula - Strings. Argumentos de 'main'.

Programação
Mestrado em Engenharia Física Tecnológica

Samuel M. Eleutério
sme@tecnico.ulisboa.pt

Departamento de Física
Instituto Superior Técnico
Universidade de Lisboa

Strings - Introdução

- A utilização frequente de **sequências de caracteres** (texto, por exemplo) conduz à necessidade de **tipos especificamente** orientados para o seu tratamento adequado.
- Na sua forma mais simples, associamos **um byte** a cada **caracter** (**letra**) e designamos o tipo associado por **char**. Assim, uma **frase** (ou um **texto**) não é mais do que um **vetor de caracteres** (variável dimensionada), ou seja, um **vetor de char**.
- Para declararmos uma variável como **char**:

  char letra = 'a';

  Note-se a utilização das **plicas**. De facto, um **char** comporta-se como um **inteiro** com **256** valores possíveis (2^8). Quando se colocam as plicas estamos a **atribuir à variável** o **valor numérico** da letra **a** (isto é, o seu código **ASCII**, **97**).
- Ver tabela ASCII anexa ou nos sites: '**www.tabelaascii.com**', '**pt.wikipedia.org/wiki/ASCII**', etc.

Strings ('Prog09_01.c')

- Para **guardarmos** um conjunto de caracteres teremos de utilizar uma **variável dimensionada**:

  char texto[80] = "Isto e um texto";
  char *texto = "Isto e um texto";

  Em ambos os casos estamos a atribuir um **valor** à variável **texto**:

  - No primeiro caso a variável é declarada com **80 bytes**;
  - No segundo é reservado apenas o **espaço necessário** ao texto.

- Mas, como sabemos quando o texto termina nestas **sequências de caracteres** (**strings**)?
- Em **C**, por convenção, as **strings** (recorde-se, sequências de caracteres) são terminadas pelo caracter '0' do código **ASCII**. **Não confundir com o número '0' cujo código é o '48'**.
- Assim, no exemplo anterior, terão de ser reservados **15 bytes** para texto e **um byte** para o '0' final, ou seja **16 bytes**.

Vetores de Strings ('Prog09_02.c')

- Uma **string** é uma **sequência de caracteres** (**vetor de char**);
- Se quisermos agora ter um vetor de strings temos de ter uma variável dimensionada ('**vs**') com duas dimensões associadas:

0	l	s	t	o		e		u	m		t	e	x	t	o	ϕ
1	q	u	e		s	e	r	v	e		p	a	r	a	ϕ	
2	e	x	e	m	p	l	i	f	i	c	a	r	ϕ			
3	c	o	m	o		e		o		v	e	t	o	r	ϕ	
4	d	e		s	t	r	i	n	g	s	.	ϕ				

- Aqui, **cada linha** mais não é do que a linha uma **matriz**. Deste modo, o valor de **vs[i]** é o ponteiro para vetor '**i**'. Por exemplo, **vs[2]** é o ponteiro que aponta para o terceiro vetor. Note-se que o comprimento mínimo será de **16** bytes (**15** de **texto** + **1** do '**0**').
- Assim, temos um **vetor** de '0' até '4', em que, cada um dos seus elementos é um vetor '**char**'!

Vetores de Strings ('Prog09_02.c')

- A declaração dos vetores de texto anteriores, '**vs**', será então

  char vs[5][16] ;

  em que temos **5** vetores de **16** bytes.
- Quando passamos esta variável para uma função ('**func**'), *ela não sabe* como se encontra estruturada aquela zona de memória, por isso, temos de **indicar** que ela se organiza em grupos de **16** bytes:

  tipo func (**char vs[][16]**, '**outros_argumentos**') {...}

- Um outro modo de se definirem os **vetores de vetores** é a partir dos seus **ponteiros**.
- À primeira vista os dois processos **parecem idênticos**, tanto mais que as variáveis por eles definidas são usados da mesma maneira.
- A sua **distinção** está na sua **definição** e consequentemente no modo como são **transferidos** para funções.

Vetores de Strings ('Prog09_03.c')

- Se quisermos manter os textos do nosso exemplo, podemos definir separadamente os **5 vetores de texto** ('**strings**'):

  char ch0[16] = "Isto e um texto";

  char ch1[15] = "que serve para"; etc..

- Depois criamos um **vetor de ponteiros para char**:

  char *vs[5] ;

- E finalmente **preenchemos** cada um dos 5 elementos deste com os ponteiros para as strings previamente definidas:

  vs[0] = **ch0**; **vs[1]** = **ch1**; ...

- A partir daqui, a utilização é semelhante à do caso anterior, excepto quando passamos os **argumentos para funções**.

- Quando uma **função** receber '**vs**' ele é apenas um **vetor de ponteiros**, logo, no cabeçalho escrevemos:

  tipo func (**char **vs**, '**outros_argumentos**') {...}

Argumentos da Função 'main' ('Prog10_01.c')

- Quando falámos da função '**main**' vimos que retornava um **inteiro** e que podia **ter argumentos**.
- Uma vez que é pela função '**main**' que um programa começa, deverá ser ela a **receber** indicações iniciais do **utilizador**.
- Consideremos os dois primeiros os argumentos da função '**main**': um **vetor de ponteiros** para **strings** (2º argumento) e o **número de elementos** desse **vetor** (1º argumento):

  int main (**int argc**, **char** ****argv**) { ... }

  Nota: também se pode escrever '***argv[]**'.
- O **primeiro elemento** de '**argv**' é o comando usado para correr o programa. Os restantes correspondem ao **texto escrito a seguir**. Os **argumentos** são separados por **espaços** (ou **tabs**):

  ./Prog10_01 a b c
- Ver alteração do programa **Prog08_04**: **Prog08_06**.