9ª Aula - Manipulação de '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.

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**.

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 **vector de caracteres** (variável dimensionada), ou seja, um **vector de char**.

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 **vector de caracteres** (variável dimensionada), ou seja, um **vector de char**.
- Para declararmos uma variável como **char**:

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 **vector de caracteres** (variável dimensionada), ou seja, um **vector de char**.
- Para declararmos uma variável como **char**:

  char letra = 'a';

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 **vector de caracteres** (variável dimensionada), ou seja, um **vector de char**.
- Para declararmos uma variável como **char**:

  char letra = 'a';

  Note-se a utilização das **plicas**. De facto, um **char** é um **inteiro** de 1 byte, logo, com **256** valores possíveis (2^8).

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 **vector de caracteres** (variável dimensionada), ou seja, um **vector de char**.
- Para declararmos uma variável como **char**:

  char letra = 'a';

  Note-se a utilização das **plicas**. De facto, um **char** é um **inteiro** de 1 byte, logo, 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**).

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 **vector de caracteres** (variável dimensionada), ou seja, um **vector de char**.
- Para declararmos uma variável como **char**:

  char letra = 'a';

  Note-se a utilização das **plicas**. De facto, um **char** é um **inteiro** de 1 byte, logo, 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, por exemplo, no site da Wikipedia:

  https://en.wikipedia.org/wiki/ASCII

Strings ('Prog09_01e2.c')

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

Strings ('Prog09_01e2.c')

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

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

Strings ('Prog09_01e2.c')

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

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

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

Strings ('Prog09_01e2.c')

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

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

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

  - No primeiro caso a variável é declarada com **80 bytes**;

Strings ('Prog09_01e2.c')

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

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

Strings ('Prog09_01e2.c')

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

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

Strings ('Prog09_01e2.c')

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

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

Strings ('Prog09_01e2.c')

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

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

Strings ('Prog09_01e2.c')

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

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

Vectores de Strings ('Prog09_03.c')

- Uma **string** é uma **sequência de caracteres** (**vector de char**);

Vectores de Strings ('Prog09_03.c')

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

Vectores de Strings ('Prog09_03.c')

- Uma **string** é uma **sequência de caracteres** (**vector de char**);
- Se quisermos agora ter um vector 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				

Vectores de Strings ('Prog09_03.c')

- Uma **string** é uma **sequência de caracteres** (**vector de char**);
- Se quisermos agora ter um vector 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				

- Assim, **cada linha** mais não é do que uma linha duma **matriz**. Deste modo, o valor de **vs[i]** é o ponteiro para vector '**i**'.

Vectores de Strings ('Prog09_03.c')

- Uma **string** é uma **sequência de caracteres** (**vector de char**);
- Se quisermos agora ter um vector 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				

- Assim, **cada linha** mais não é do que uma linha duma **matriz**. Deste modo, o valor de **vs[i]** é o ponteiro para vector '**i**'. Por exemplo, **vs[2]** é o ponteiro que aponta para o terceiro vector.

Vectores de Strings ('Prog09_03.c')

- Uma **string** é uma **sequência de caracteres** (**vector de char**);
- Se quisermos agora ter um vector 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				

- Assim, **cada linha** mais não é do que uma linha duma **matriz**. Deste modo, o valor de **vs[i]** é o ponteiro para vector '**i**'. Por exemplo, **vs[2]** é o ponteiro que aponta para o terceiro vector. Note-se que o comprimento mínimo será de **16** bytes (**15** de **texto** + **1** do '**0**').

Vectores de Strings ('Prog09_03.c')

- Uma **string** é uma **sequência de caracteres** (**vector de char**);
- Se quisermos agora ter um vector 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				

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

Vectores de Strings ('Prog09_03.c')

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

  char vs[5][16] ;

  em que temos **5** vectores de **16** bytes.

Vectores de Strings ('Prog09_03.c')

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

  char vs[5][16] ;

  em que temos **5** vectores 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:

Vectores de Strings ('Prog09_03.c')

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

  char vs[5][16] ;

  em que temos **5** vectores 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**') {...}

Vectores de Strings ('Prog09_03.c')

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

  char vs[5][16] ;

  em que temos **5** vectores 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 **vectores de vectores** é a partir dos seus **ponteiros**.

Vectores de Strings ('Prog09_03.c')

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

  char vs[5][16] ;

  em que temos **5** vectores 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 **vectores de vectores** 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.

Vectores de Strings ('Prog09_03.c')

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

  char vs[5][16] ;

  em que temos **5** vectores 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 **vectores de vectores** 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.

Vectores de Strings ('Prog09_04/05.c')

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

Vectores de Strings ('Prog09_04/05.c')

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

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

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

Vectores de Strings ('Prog09_04/05.c')

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

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

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

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

  char *vs[5] ;

Vectores de Strings ('Prog09_04/05.c')

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

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

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

- Depois criamos um **vector 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**; ...

Vectores de Strings ('Prog09_04/05.c')

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

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

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

- Depois criamos um **vector 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**.

Vectores de Strings ('Prog09_04/05.c')

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

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

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

- Depois criamos um **vector 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 **vector de ponteiros**, logo, no cabeçalho escrevemos:

Vectores de Strings ('Prog09_04/05.c')

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

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

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

- Depois criamos um **vector 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**; ...

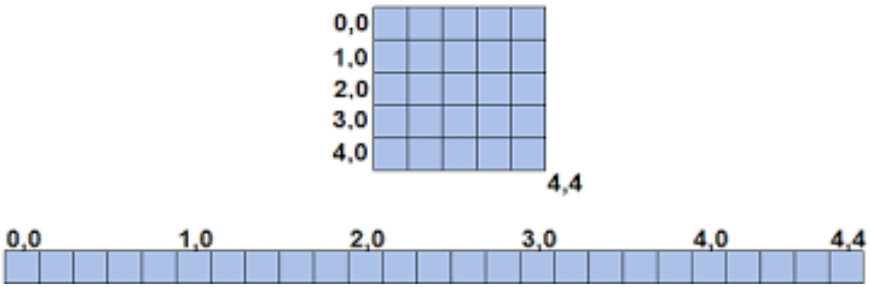
- A partir daqui, a utilização é semelhante à do caso anterior, excepto quando passamos os **argumentos para funções**.

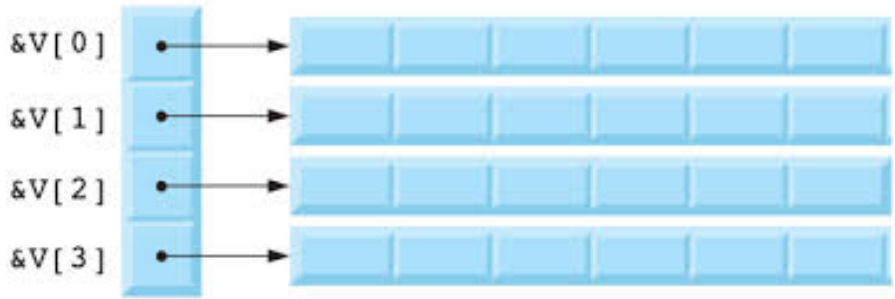
- Quando uma **função** receber '**vs**' ele é apenas um **vector de ponteiros**, logo, no cabeçalho escrevemos:

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

Funções para manipular 'Strings'

- Na figura do lado esquerdo mostra-se um vector a duas dimensões. Note-se que o espaço reservado é contínuo e dividido em partes iguais.
- Na figura do lado direito, mostra-se um vector de ponteiros para vectores. Apesar da figura mostrar todos os vectores iguais, eles podem ter tamanhos diferentes.





Nota: Imagens tiradas da net. Não havia referências aos seus autores.

Funções para manipular 'Strings' ('Prog11_01.c')

- À semelhança das funções '**scanf**' para ler do terminal, ou '**fscanf**' para ler duma file, existe a função '**sscanf**' para a leitura de uma variável a partir de uma **string** ('**s**'):

Funções para manipular 'Strings' ('Prog11_01.c')

- À semelhança das funções '**scanf**' para ler do terminal, ou '**fscanf**' para ler duma file, existe a função '**sscanf**' para a leitura de uma variável a partir de uma **string** ('**s**'):

  int sscanf (const char *s, const char *template, ...)

Funções para manipular 'Strings' ('Prog11_01.c')

- À semelhança das funções '**scanf**' para ler do terminal, ou '**fscanf**' para ler duma file, existe a função '**sscanf**' para a leitura de uma variável a partir de uma **string** ('**s**'):

  int sscanf (const char *s, const char *template, ...)

- Se desejarmos ler a partir da **string**:

Funções para manipular 'Strings' ('Prog11_01.c')

- À semelhança das funções '**scanf**' para ler do terminal, ou '**fscanf**' para ler duma file, existe a função '**sscanf**' para a leitura de uma variável a partir de uma **string** ('**s**'):

  int sscanf (**const char** ***s**, **const char** ***template**, ...)

- Se desejarmos ler a partir da **string**:

  char texto[64] = "12 15 7.3" ;

Funções para manipular 'Strings' ('Prog11_01.c')

- À semelhança das funções '**scanf**' para ler do terminal, ou '**fscanf**' para ler duma file, existe a função '**sscanf**' para a leitura de uma variável a partir de uma **string** ('**s**'):

  int sscanf (**const char** ***s**, **const char** ***template**, ...)

- Se desejarmos ler a partir da **string**:

  char texto[64] = "12 15 7.3" ;

  teremos simplesmente de fazer:

Funções para manipular 'Strings' ('Prog11_01.c')

- À semelhança das funções '**scanf**' para ler do terminal, ou '**fscanf**' para ler duma file, existe a função '**sscanf**' para a leitura de uma variável a partir de uma **string** ('**s**'):

  int sscanf (**const char** ***s**, **const char** ***template**, ...)

- Se desejarmos ler a partir da **string**:

  char texto[64] = "12 15 7.3" ;

  teremos simplesmente de fazer:

  sscanf (**texto**, "**%d %d %f**", **&i1**, **&i2**, **&x3**);

Funções para manipular 'Strings' ('Prog11_01.c')

- À semelhança das funções '**scanf**' para ler do terminal, ou '**fscanf**' para ler duma file, existe a função '**sscanf**' para a leitura de uma variável a partir de uma **string** ('**s**'):

  int sscanf (**const char** ***s**, **const char** ***template**, ...)

- Se desejarmos ler a partir da **string**:

  char texto[64] = "12 15 7.3" ;

  teremos simplesmente de fazer:

  sscanf (**texto**, "**%d %d %f**", **&i1**, **&i2**, **&x3**);

- Inversamente, se quisermos escrever numa **string**, podemos usar uma função análoga à '**printf**' ou à '**fprintf**' que se chama '**sprintf**' e tem uma sintaxe idêntica à das anteriores:

Funções para manipular 'Strings' ('Prog11_01.c')

- À semelhança das funções '**scanf**' para ler do terminal, ou '**fscanf**' para ler duma file, existe a função '**sscanf**' para a leitura de uma variável a partir de uma **string** ('**s**'):

  int sscanf (**const char** ***s**, **const char** ***template**, ...)

- Se desejarmos ler a partir da **string**:

  char texto[64] = "12 15 7.3" ;

  teremos simplesmente de fazer:

  sscanf (**texto**, "**%d %d %f**", **&i1**, **&i2**, **&x3**);

- Inversamente, se quisermos escrever numa **string**, podemos usar uma função análoga à '**printf**' ou à '**fprintf**' que se chama '**sprintf**' e tem uma sintaxe idêntica à das anteriores:

  int sprintf (**const char** ***s**, **const char** ***template**, ...)

Funções para manipular 'Strings' ('Prog11_01.c')

- À semelhança das funções '**scanf**' para ler do terminal, ou '**fscanf**' para ler duma file, existe a função '**sscanf**' para a leitura de uma variável a partir de uma **string** ('**s**'):

  int sscanf (**const char** ***s**, **const char** ***template**, ...)

- Se desejarmos ler a partir da **string**:

  char texto[64] = "12 15 7.3" ;

  teremos simplesmente de fazer:

  sscanf (**texto**, "**%d %d %f**", **&i1**, **&i2**, **&x3**);

- Inversamente, se quisermos escrever numa **string**, podemos usar uma função análoga à '**printf**' ou à '**fprintf**' que se chama '**sprintf**' e tem uma sintaxe idêntica à das anteriores:

  int sprintf (**const char** ***s**, **const char** ***template**, ...)

- Podemos assim escrever numa **string**:

Funções para manipular 'Strings' ('Prog11_01.c')

- À semelhança das funções '**scanf**' para ler do terminal, ou '**fscanf**' para ler duma file, existe a função '**sscanf**' para a leitura de uma variável a partir de uma **string** ('**s**'):

  int sscanf (**const char** ***s**, **const char** ***template**, ...)

- Se desejarmos ler a partir da **string**:

  char texto[64] = "12 15 7.3" ;

  teremos simplesmente de fazer:

  sscanf (**texto**, "**%d %d %f**", **&i1**, **&i2**, **&x3**);

- Inversamente, se quisermos escrever numa **string**, podemos usar uma função análoga à '**printf**' ou à '**fprintf**' que se chama '**sprintf**' e tem uma sintaxe idêntica à das anteriores:

  int sprintf (**const char** ***s**, **const char** ***template**, ...)

- Podemos assim escrever numa **string**:

  sprintf (**texto**, "Estão aqui **%d** pessoas.\n", **i1**);

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

- Quando falámos da função '**main**' vimos que retornava um **inteiro** e que podia **ter 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**.

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 **vector de ponteiros** para **strings** (2º argumento) e o **número de elementos** desse **vector** (1º argumento):

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 **vector de ponteiros** para **strings** (2º argumento) e o **número de elementos** desse **vector** (1º argumento):

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

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 **vector de ponteiros** para **strings** (2º argumento) e o **número de elementos** desse **vector** (1º argumento):

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

  Nota: também se pode escrever '***argv[]**'.

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 **vector de ponteiros** para **strings** (2º argumento) e o **número de elementos** desse **vector** (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.

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 **vector de ponteiros** para **strings** (2º argumento) e o **número de elementos** desse **vector** (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**.

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 **vector de ponteiros** para **strings** (2º argumento) e o **número de elementos** desse **vector** (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**:

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 **vector de ponteiros** para **strings** (2º argumento) e o **número de elementos** desse **vector** (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**:

  ./Prog10_01 a b c
- Ver alteração do programa '**Prog08_07.c**': '**Prog08_10.c**'.

Funções para manipular 'Strings' ('Prog11_02a6.c')

- O **C** dispõe de um conjunto de **funções** para manipular **strings**. As suas **definições** encontram-se '**string.h**'.

Funções para manipular 'Strings' ('Prog11_02a6.c')

- O **C** dispõe de um conjunto de **funções** para manipular **strings**. As suas **definições** encontram-se '**string.h**'.
- Como primeira abordagem da manipulação de **strings** vejamos, por alto, algumas desses funções:

Funções para manipular 'Strings' ('Prog11_02a6.c')

- O **C** dispõe de um conjunto de **funções** para manipular **strings**. As suas **definições** encontram-se '**string.h**'.
- Como primeira abordagem da manipulação de **strings** vejamos, por alto, algumas desses funções:
- Para **copiar** strings (o destino tem de ter espaço suficiente):

```
char *strcpy (char *destino, char *origem);
```

Funções para manipular 'Strings' ('Prog11_02a6.c')

- O **C** dispõe de um conjunto de **funções** para manipular **strings**. As suas **definições** encontram-se '**string.h**'.
- Como primeira abordagem da manipulação de **strings** vejamos, por alto, algumas desses funções:
- Para **copiar** strings (o destino tem de ter espaço suficiente):

  char ***strcpy** (**char** ***destino**, **char** ***origem**);
- Para **juntar** no final (o destino tem de ter espaço suficiente):

  char ***strcat** (**char** ***destino**, **char** ***origem**);

Funções para manipular 'Strings' ('Prog11_02a6.c')

- O **C** dispõe de um conjunto de **funções** para manipular **strings**. As suas **definições** encontram-se '**string.h**'.
- Como primeira abordagem da manipulação de **strings** vejamos, por alto, algumas desses funções:
- Para **copiar** strings (o destino tem de ter espaço suficiente):

  char ***strcpy** (**char** ***destino**, **char** ***origem**);
- Para **juntar** no final (o destino tem de ter espaço suficiente):

  char ***strcat** (**char** ***destino**, **char** ***origem**);
- Para calcular o **tamanho** de uma string, isto é, a posição do caracter **ASCII** '**0**' (**size_t** é um inteiro positivo):

  size_t strlen (**char** ***str**);

Funções para manipular 'Strings' ('Prog11_02a6.c')

- O **C** dispõe de um conjunto de **funções** para manipular **strings**. As suas **definições** encontram-se '**string.h**'.
- Como primeira abordagem da manipulação de **strings** vejamos, por alto, algumas desses funções:
- Para **copiar** strings (o destino tem de ter espaço suficiente):

  char ***strcpy** (**char** ***destino**, **char** ***origem**);
- Para **juntar** no final (o destino tem de ter espaço suficiente):

  char ***strcat** (**char** ***destino**, **char** ***origem**);
- Para calcular o **tamanho** de uma string, isto é, a posição do caracter **ASCII** '**0**' (**size_t** é um inteiro positivo):

  size_t strlen (**char** ***str**);
- Para **comparar** duas strings (retorna '**0**' se iguais, '**negativo**' se 'str2' fôr maior do que 'str1' e '**positivo**' no caso contrário):

Funções para manipular 'Strings' ('Prog11_02a6.c')

- O **C** dispõe de um conjunto de **funções** para manipular **strings**. As suas **definições** encontram-se '**string.h**'.
- Como primeira abordagem da manipulação de **strings** vejamos, por alto, algumas desses funções:
- Para **copiar** strings (o destino tem de ter espaço suficiente):

  char ***strcpy** (**char** ***destino**, **char** ***origem**);
- Para **juntar** no final (o destino tem de ter espaço suficiente):

  char ***strcat** (**char** ***destino**, **char** ***origem**);
- Para calcular o **tamanho** de uma string, isto é, a posição do caracter **ASCII** '**0**' (**size_t** é um inteiro positivo):

  size_t strlen (**char** ***str**);
- Para **comparar** duas strings (retorna '**0**' se iguais, '**negativo**' se 'str2' fôr maior do que 'str1' e '**positivo**' no caso contrário):

  int strcmp (**char** ***str1**, **char** ***str2**);