AULA PRÁTICA N.º 4

# Objetivos:

* Manipulação de *arrays* em linguagem C, usando índices e ponteiros.
* Tradução para *assembly* de código de acesso sequencial a *arrays* usando índices e ponteiros. Parte 1.

# Guião:

1. O programa seguinte lê uma *string* do teclado, conta o número de carateres numéricos que ela contém e imprime esse resultado.

**#define SIZE 20**

**void main (void)**

**{**

**static char str[SIZE+1]; // Reserva espaço para um array de**

**//"SIZE+1" bytes no segmento de**

**// dados ("SIZE" carateres +**

**// terminador)**

**int num, i;**

**read\_string(str, SIZE); // "str" é o endereço inicial do**

**// espaço reservado para alojar a**

**// string (na memória externa)**

**num = 0;**

**i = 0;**

**while( str[i] != '\0' ) // Acede ao carater (byte) na**

**// posição "i" do array e compara-o**

**// com o carater terminador (i.e.**

**// '\0' = 0x00)**

**{**

**if( (str[i] >= '0') && (str[i] <= '9') ) num++;**

**i++;**

**}**

**print\_int10(num);**

**}**

* 1. Codifique o programa em *assembly* do MIPS e teste o seu funcionamento no MARS. Utilize, para armazenar as variáveis, os seguintes registos: **num** (**$t0**), **i** (**$t1**), endereço inicial da *string* (**$t2**), endereço da posição "**i**" da *string* (**$t3**) e conteúdo de **str[i]** (**$t4**).

Tradução parcial do código anterior para *assembly*:

**# Mapa de registos # num: $t0**

**# i: $t1**

**# str: $t2 # str+i: $t3 # str[i]: $t4**

**.data**

**.eqv SIZE,20**

**.eqv read\_string,...**

**.eqv print\_int10,... str: .space ...**

|  |  |  |  |
| --- | --- | --- | --- |
| **.text** |  |  | |
| **.globl** | **main** |
| **main: la** | **$a0,...** | **#** | **$a0=&str[0] (endereço da posição** |
|  |  | **#** | **0 do array, i.e., endereço** |
|  |  | **#** | **inicial do array)** |
| **li** | **$a1,...** | **#** | **$a1=SIZE** |

**li $v0,read\_string**

**syscall # read\_string(str,SIZE)**

**(...) # num=0; i=0;**

**while: # while(str[i] != '\0') la $t2,str # $t2 = str ou &str[0] addu $t3,... # $t3 = str+i ou &str[i] lb $t4,0(...) # $t4 = str[i]**

**b?? $t4,'\0',endw # {**

**if: b?? $t4,'0',endif # if(str[i] >= '0' &&**

**b?? $t4,'9',endif # str[i] <= '9'); addi $t0,... # num++;**

**endif:**

**addi $t1,... # i++;**

**j ... # }**

**endw: (...) # print\_int10(num); jr $ra # termina o programa**

* 1. Execute o programa passo a passo, introduza a string **"AC1-Labs"** e preencha a tabela abaixo com os valores que as diferentes variáveis vão tomando:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Endereço de**  **str ($t2)** | **Endereço de**  **str[i]($t3)** | **str[i]**  **($t4)** | **i**  **($t1)** | **num**  **($t0)** |  |
|  |  |  | **0** | **0** | **Val. iniciais** |
| 268500992 | 26850992 | 64 | 1 | 0 | **Fim 1ª iter.** |
| 268500992 | 268500993 | 67 | 2 | 0 | **Fim 2ª iter.** |
| 268500992 | 268500994 | 49 | 3 | 1 | **Fim 3ª iter.** |
| 268500992 | 268500995 | 45 | 4 | 1 | **Fim 4ª iter.** |
| 268500992 | 268500996 | 108 | 5 | 1 | **Fim 5ª iter.** |
| 268500992 | 268500997 | 97 | 6 | 1 | **Fim 6ª iter.** |
| 268500992 | 268500998 | 98 | 7 | 1 | **Fim 7ª iter.** |
| 268500992 | 268500999 | 115 | 8 | 1 | **Fim 8ª iter.** |

1. Uma forma alternativa de escrever o código da questão 1 consiste na utilização de um ponteiro para aceder a cada um dos elementos do *array*. O ponteiro para uma dada posição do *array* é uma variável (que pode residir num registo interno do CPU) que contém o endereço dessa posição do *array*. Se, inicialmente, for atribuído a esse ponteiro o endereço da primeira posição do *array*, para efetuar o acesso sequencial a cada uma das posições restantes é necessário incrementar sucessivamente o valor do ponteiro.

A implementação do programa da questão 1 usando ponteiros é apresentada de seguida:

**#define SIZE 20**

**void main (void)**

**{**

**static char str[SIZE+1]; // Reserva espaço para um array de**

**// "SIZE+1" carateres no segmento de**

**//dados**

**int num = 0;**

**char \*p; // Declara um ponteiro para carater**

**// (não há qualquer inicialização) read\_string(str, SIZE); // Le do teclado uma string com um**

**// máximo de 20 carateres**

**p = str; // Inicializa o ponteiro "p" com o**

**// endereço inicial da string**

**// (equivalente a p = &(str[0])) while( \*p != '\0' ) // Acede ao byte apontado pelo**

**// ponteiro "p" (\*p) e compara**

**// o valor lido com o carater**

**// terminador ('\0' = 0x00)**

**{**

**if( (\*p >= '0') && (\*p <= '9') ) num++;**

**p++; // Incrementa o ponteiro (o ponteiro**

**// passa a ter o endereço da**

**// posição seguinte do array)**

**}**

**print\_int10(num);**

**}**

* 1. Codifique o programa em *assembly* do MIPS e teste o seu funcionamento no MARS. Utilize, para armazenar as variáveis, os seguintes registos: **$t0** (**num**), **$t1** (**p**), **$t2** (**\*p**). Tradução parcial do código anterior para *assembly*:

**# Mapa de registos # num: $t0**

**# p: $t1**

|  |  |  |  |
| --- | --- | --- | --- |
| **# \*p: $t2** |  |  | |
| **.data** |  |
| **...** |  |
| **.text** |  |
| **.globl** | **main** |
| **main: ...** |  | **#** | **...** |
| **la** | **$t1,str** | **#** | **p = str;** |
| **while:** |  | **#** | **while(\*p != '\0')** |
| **lb** | **$t2,...** | **#** |  |
| **b??** | **$t2,0,endw** | **#** | **{** |
| **b??** | **$t2,'0',endif** | **#** | **if(str[i] >='0' &&** |
| **b??** | **$t2,'9',endif** | **#** | **str[i] <= '9')** |
| **addi** | **$t0,...** | **#** | **num++;** |
| **endif:** |  |  |  |
| **addiu** | **$t1,...** | **#** | **p++;** |
| **(...)** |  | **#** | **}** |
| **endw: (...)** |  | **#** | **print\_int10(num);** |
| **jr** | **$ra** | **#** | **termina o programa** |

* 1. Execute o programa passo a passo, introduza a string **"AC1-Labs"** e preencha a tabela abaixo com os valores que as diferentes variáveis vão tomando:

|  |  |  |  |
| --- | --- | --- | --- |
| **num ($t0)** | **p ($t1)** | **\*p ($t2)** |  |
|  |  |  | **Valores iniciais** |
|  |  |  | **Fim da 1ª iteração** |
|  |  |  | **Fim da 2ª iteração** |
|  |  |  | **Fim da 3ª iteração** |
|  |  |  | **Fim da 4ª iteração** |
|  |  |  | **Fim da 5ª iteração** |
|  |  |  | **Fim da 6ª iteração** |
|  |  |  | **Fim da 7ª iteração** |
|  |  |  | **Fim da 8ª iteração** |

1. O programa seguinte calcula e imprime a soma dos elementos de um *array* de 4 posições. Esta implementação utiliza um ponteiro para aceder sucessivamente a cada uma das posições do *array* ("**p**") e um outro ponteiro, que atua como uma constante, para indicar o endereço da última posição do *array* de inteiros (ao contrário de uma *string*, um *array* de inteiros não possui qualquer elemento que indique terminação).

**#define SIZE 4**

**int array[4] = {7692, 23, 5, 234}; // Declara um array global de 4**

**// posições e inicializa-o**

**void main (void)**

**{**

**int \*p; // Declara um ponteiro para inteiro**

**// (não há qualquer inicialização) int \*pultimo; // Declara um ponteiro para inteiro int soma = 0;**

**p = array; // "p" é preenchido com o endereço**

**// inicial do array pultimo=array+SIZE-1; // "pultimo" é inicializado com o**

**// endereço do último elemento do**

**// array, i.e., &array[SIZE-1]**

**while( p <= pultimo )**

**{**

**soma = soma + (\*p);**

**p++; // Incrementa o ponteiro (não esquecer**

**// que incrementar um ponteiro para um**

**// inteiro de 32 bits significa somar a**

**// quantidade 4 ao valor do endereço)**

**}**

**print\_int10(soma);**

**}**

* 1. Codifique o programa em *assembly* do MIPS e teste o seu funcionamento no MARS. Utilize, para armazenar as variáveis, os seguintes registos: **$t0** (**p**), **$t1** (**pultimo**), **$t2** (**\*p**), **$t3** (**soma**).

Tradução parcial do código anterior para *assembly*:

**# Mapa de registos # p: $t0**

**# pultimo:$t1 # \*p $t2**

**# soma: $t3**

**.data**

|  |  |  |
| --- | --- | --- |
| **array:.word** | **7692,23,...** |  |
| **.eqv** | **print\_int10,...** |
| **.eqv** | **SIZE,4** |
| **.text** |  |
| **.globl** | **main** |
| **main: li** | **$t3,.. #** | **soma = 0;** |
| **li** | **$t4,SIZE #** |  |
| **sub** | **$t4,$t4,1 #** | **$t4 = 3** |
| **sll** | **$t4,$t4,2 #** | **ou "mul $t4,$t4,4"** |
| **la** | **$t0,... #** | **p = array;** |
| **addu** | **$t1,$t0,... #** | **pultimo = array + SIZE - 1;** |
| **while:** | **#** | **while(p <= pultimo)** |
| **b??u** | **$t0,...,endw #** | **{** |
| **...** | **$t2,0(...) #** | **$t2 = \*p;** |
| **add** | **$t3,... #** | **soma = soma + (\*p);** |
| **addiu** | **$t0,$t0,... #** | **p++;** |
| **(...)** | **#** | **}** |
| **(...)** | **#** | **print\_int10(soma);** |
| **jr** | **$ra #** | **termina o programa** |

* 1. Execute o programa passo a passo e preencha a tabela abaixo com os valores que as diferentes variáveis vão tomando:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **p ($t0)** | **pultimo ($t1)** | **\*p ($t2)** | **soma ($t3)** |  |
|  |  |  |  | **Valores iniciais** |
|  |  |  |  | **Fim 1ª iteração** |
|  |  |  |  | **Fim 2ª iteração** |
|  |  |  |  | **Fim 3ª iteração** |
|  |  |  |  | **Fim 4ª iteração** |

* 1. Altere o programa em C de modo a utilizar o acesso ao *array* com índices. Faça as alterações correspondentes ao programa *assembly* e teste o seu funcionamento no MARS.

# Exercícios adicionais

1. Considere o seguinte programa que lê da consola uma *string* com um máximo de 20 carateres, converte, de forma parcialmente correta, os carateres correspondentes a letras minúsculas em maiúsculas e, por fim, escreve a *string* alterada no ecrã.

**#define SIZE 20**

**void main(void)**

**{**

**static char str[SIZE+1]; char \*p;**

**print\_string("Introduza uma string: "); read\_string(str, SIZE);**

**p = str;**

**while (\*p != '\0')**

**{**

**\*p = \*p – 'a' + 'A'; // 'a'=0x61, 'A'=0x41, 'a'-'A'=0x20 p++;**

**}**

**print\_string(str);**

**}**

* 1. Codifique o programa em *assembly* do MIPS e teste o seu funcionamento no MARS. Utilize, para armazenar as variáveis, os seguintes registos: **p** (**$t0**), \***p** (**$t1**).
  2. Execute o programa passo a passo, introduza a string **"Ac1-prAticaS"** e preencha a tabela abaixo com os valores que as diferentes variáveis vão tomando:

|  |  |  |
| --- | --- | --- |
| **p ($t0)** | **\*p ($t1)** |  |
|  |  | **Valores iniciais** |
|  |  | **Fim da 1ª iteração** |
|  |  | **Fim da 2ª iteração** |
|  |  | **Fim da 3ª iteração** |
|  |  | **Fim da 4ª iteração** |
|  |  | **Fim da 5ª iteração** |
|  |  | **Fim da 6ª iteração** |
|  |  | **Fim da 7ª iteração** |
|  |  | **Fim da 8ª iteração** |
|  |  | **Fim da 9ª iteração** |
|  |  | **Fim da 10ª iteração** |
|  |  | **Fim da 11ª iteração** |
|  |  | **Fim da 12ª iteração** |

* 1. Como pôde verificar, o programa anterior apenas produz o resultado esperado em alguns casos. Proponha uma alteração ao programa para corrigir o problema detetado, codifique- a em *assembly* e teste-a no MARS.
  2. Altere o programa em C resultante do ponto anterior de modo a converter letras maiúsculas em minúsculas. Faça a correspondente alteração do programa *assembly* e teste o seu funcionamento.

# Anexo:

Interpretação gráfica de ponteiros (supondo uma máquina de 32 bits)

## Ponteiro para carater, não inicializado

* 1. Exemplo de declaração em linguagem C:

**char \*p;**

* 1. Interpretação gráfica:

**32 bits 8 bits**

**?**

**(address)**

**?**

**(value)**

**(char)**

* 1. Ação desenvolvida na tradução para linguagem máquina:
     + Definir o registo interno / reservar espaço na memória para alojar um endereço (32 bits)
  2. Caso o ponteiro resida num registo interno, basta definir qual o registo a usar para esse efeito e incluí-lo nas instruções que manipulam o ponteiro.
  3. Caso o ponteiro resida na memória, uma possível tradução para *Assembly* do MIPS da sua declaração é:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **ptr\_p:** | **.space** | **4** | **#** | **Reserva 4** | **bytes de memória** |
|  | | | **#** | **(32 bits)** | **para alojar o** |
| **#** | **ponteiro.** | **Não há inicialização** |

## Ponteiro para inteiro, não inicializado

* 1. Exemplo de declaração em linguagem C:

**int \*p;**

* 1. Interpretação gráfica:

**32 bits 32 bits**

**?**

**(address)**

**?**

**(value)**

**(int)**

* 1. Ação desenvolvida na tradução para linguagem máquina:
     + Reservar espaço na memória/registo interno para um endereço (32 bits)
  2. Possível tradução para *Assembly* do MIPS (caso o ponteiro resida na memória):

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **ptr\_p:** | **.space** | **4** | **#** | **Reserva 4** | **bytes de memória** |
|  | | | **#** | **(32 bits)** | **para alojar o** |
| **#** | **ponteiro.** | **Não há inicialização** |

1. ***Array* de carateres**
   1. Exemplo de declaração em linguagem C:

**char p[]="OLA";**

* 1. Interpretação gráfica:

**8 bits**

|  |  |  |  |
| --- | --- | --- | --- |
| **'O'** | **'L'** | **'A'** | **'\0'** |

**p**

**8 bits**

**8 bits**

**8 bits**

**Terminador (0x00)**

**(char)**

**(char)**

**(char)**

**(char)**

* 1. Ação desenvolvida na tradução para linguagem máquina:
     + Reservar espaço na memória para um *array* de carateres (incluindo para o terminador, o byte 0x00), e efetuar a respetiva inicialização
  2. Possível tradução para *Assembly* do MIPS:

**p: .asciiz "OLA" # Reserva 4 bytes de memória e**

**# inicializa-os com os códigos # ASCII dos 3 carateres e com o # código do terminador (0).**

**# O valor de "p" pode ser obtido # com a instrução "load address"**

Ou, alternativamente:

**p: .ascii "OLA" # Reserva 3 bytes de memória e**

**# inicializa-os com os códigos # ASCII dos 3 carateres**

**.byte 0x00 # Reserva 1 byte e inicializa-o # com o valor 0**

## Ponteiro para *Array* de carateres

* 1. Exemplo de declaração em linguagem C:

**char \*p = "OLA";**

* 1. Interpretação gráfica:

**8 bits**

**32 bits**

**p**

**(address)**

**8 bits**

**8 bits**

**8 bits**

|  |  |  |  |
| --- | --- | --- | --- |
| **'O'** | **'L'** | **'A'** | **'\0'** |

**(char)**

**(char)**

**(char)**

**(char)**

* 1. Ações desenvolvidas na tradução para linguagem máquina:
     + Reservar espaço para um *array* de carateres e efetuar a respetiva inicialização
     + Reservar espaço para um endereço e efetuar a respetiva inicialização
  2. Tradução para *Assembly* do MIPS (caso o ponteiro resida na memória):

**p: .asciiz "OLA"**

**ptr\_p: .word p # Reserva 4 bytes de memória**

**# e inicializa-os com o endereço # da primeira posição do array # de carateres (i.e. &array[0]). # O valor de "ptr\_p" pode ser**

**# obtido com a instrução "load # address"**

1. ***Array* de inteiros**
   1. Exemplo de declaração em linguagem C:

**int p[] = {12, 56, 7};**

* 1. Interpretação gráfica:

**32 bits**

|  |  |  |
| --- | --- | --- |
| **12** | **56** | **7** |

**p**

**32 bits**

**32 bits**

**(int)**

**(int)**

**(int)**

* 1. Ação desenvolvida na tradução para linguagem máquina:
     + Reservar espaço para um *array* de inteiros e efetuar a respetiva inicialização
  2. Tradução para *Assembly* do MIPS:

**p: .word 12, 56, 7 #**

**# O valor de "p" pode ser obtido # com a instrução "load address"**

NOTA:

A linguagem C não permite a declaração de um ponteiro para um *array* de inteiros, cuja representação seria, por exemplo: "**int \*p = {12, 56, 7};**". Contudo, esta declaração pode ser decomposta em duas, do seguinte modo:

## int pp[] = {12, 56, 7};

**int \*p = pp;**

1. **Ponteiro para ponteiro para carater, não inicializado**
   1. Exemplo de declaração em linguagem C:

**char \*\*p;**

* 1. Interpretação gráfica:

**32 bits 32 bits 8 bits**



**?**

**?**

**(address)**

**?**

**(address)**

**?**

**(value)**

**(char)**

* 1. Ação desenvolvida na tradução para linguagem máquina:
     + Reservar espaço para um endereço (32 bits)
  2. Tradução para *Assembly* do MIPS (caso o ponteiro resida na memória):

**ptr\_p: .space 4 # Reserva 4 bytes na memória para**

**# alojar o ponteiro**

## Ponteiro para ponteiro para inteiro, não inicializado

* 1. Exemplo de declaração em linguagem C:

**int \*\*p;**

* 1. Interpretação gráfica:

**32 bits 32 bits 32 bits**



**?**

**?**

**(address)**

**?**

**(address)**

**?**

**(value)**

**(int)**

* 1. Ação desenvolvida na tradução para linguagem máquina:
     + Reservar espaço para um endereço (32 bits)
  2. Tradução para *Assembly* do MIPS (caso o ponteiro resida na memória):

**ptr\_p: .space 4 # Reserva 4 bytes na memória para**

**# alojar o ponteiro**

## Array de ponteiros para carater

* 1. Exemplo de declaração em linguagem C:

**char \*p[] = {"AC1", "LABS"};**

* 1. Interpretação gráfica:



**32 bits**

**32 bits**

**p**

**(address)**

**(address)**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **'L'** | **'A'** | **'B'** | **'S'** | **'\0'** |

|  |  |  |  |
| --- | --- | --- | --- |
| **'A'** | **'C'** | **'1'** | **'\0'** |

* 1. Ações desenvolvidas na tradução para linguagem máquina:
     + Reservar espaço para os *arrays* de carateres e efetuar a respetiva inicialização
     + Reservar espaço para o *array* de ponteiros (*array* de inteiros) e efetuar a respetiva inicialização
  2. Tradução para *Assembly* do MIPS (caso os ponteiros residam na memória):

|  |  |  |
| --- | --- | --- |
| **array1:** | **.asciiz** | **"AC1"** |
| **array2:** | **.asciiz** | **"LABS"** |
| **p:** | **.word** | **array1, array2** |