

## AULA PRÁTICA N.º 4

### Objetivos:

- Manipulação de *arrays* em linguagem C, usando índices e ponteiros<sup>1</sup>.
- 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 caracteres 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" caracteres +
                             // 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' ) // Accede 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);
}
```

- a) 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
```

<sup>1</sup> Para uma revisão sobre o uso de ponteiros, consulte:

- Os slides “Aula 4 (Complemento)” disponíveis no moodle de AC1, na secção de aulas teóricas.
- O anexo, disponível a partir da página 7 deste guião.

```

        .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,...          # $t4 = str[i]
        b??     $t4,'\0',endw    # {
if:     b??     $t4,...          # if(str[i] >= '0' &&
        b??     $t4,...          #     str[i] <= '9');
        addi    ...              #     num++;
endif:  addi    ...              #     i++;
        j      ...              # }
endw:   (...)          # print_int10(num);
        jr      $ra              # termina o programa

```

- b) 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<br>str (\$t2) | Endereço de<br>str[i] (\$t3) | str[i]<br>(\$t4) | i<br>(\$t1) | num<br>(\$t0) |               |
|---------------------------|------------------------------|------------------|-------------|---------------|---------------|
|                           |                              |                  | 0           | 0             | Val. iniciais |
|                           |                              |                  |             |               | Fim 1ª iter.  |
|                           |                              |                  |             |               | Fim 2ª iter.  |
|                           |                              |                  |             |               | Fim 3ª iter.  |
|                           |                              |                  |             |               | Fim 4ª iter.  |
|                           |                              |                  |             |               | Fim 5ª iter.  |
|                           |                              |                  |             |               | Fim 6ª iter.  |
|                           |                              |                  |             |               | Fim 7ª iter.  |
|                           |                              |                  |             |               | Fim 8ª iter.  |

2. 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" caracteres no segmento de
                             // dados

    int num = 0;
    char *p;                // Declara um ponteiro para caracter
                             // (não há qualquer inicialização)
    read_string(str, SIZE); // Le do teclado uma string com um
                             // máximo de 20 caracteres
    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 caracter
                             // 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);
}
```

- a) 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          (Registo temporário para guardar o valor
                       armazenado na posição de memória p)

(...)
main: ...              # ...
    la      $t1, str    # p = str;
while:              # while(*p != '\0')
    lb      $t2, ...    #
    b??     $t2, '\0', endw # {
    b??     $t2, ...    #     if(str[i] >='0' &&
    b??     $t2, ...    #         str[i] <= '9')
    addi    $t0, ...    #         num++;
endif:
    addiu   ...        #     p++;
    (...)   # }
endw: (...)          # print_int10(num);
    jr      $ra        # termina o programa
```

- b) 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 |

3. O programa seguinte calcula e imprime a soma dos elementos de um *array* de inteiros 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);
}
```

- a) 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
(...)
array:.word 7692,23,...
.eqv    print_int10,...
.eqv    SIZE,...
(...)
main:li    $t3,...      # soma = 0;
li      ...           # ??? = SIZE
addi    $t4,...,-1     # $t4 = SIZE-1
sll     $t4,$t4,2      # $t4 = (SIZE-1) * 4
la      ...           # p = array;
addu    $t1,...        # pultimo = array + (SIZE - 1);
while:                                     # while(p <= pultimo)
b??u    $t0,...,endw   # {
(...)   ...           #   $t2 = *p;
add     ...           #   soma = soma + (*p);
addiu   ...           #   p++;
(...)   ...           # }
(...)   ...           # print_int10(soma);
jr      $ra           # termina o programa
```

- b) 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  |

- c) 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 caracteres, converte, de forma parcialmente correta, os caracteres 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);
}
```

- a) 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)**.
- b) Execute o programa passo a passo, introduza a string "**Ac1-prÁticaS**" 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 |

- c) 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.
- d) 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:**

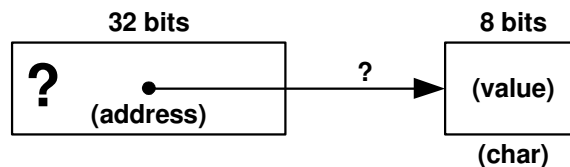
Interpretação gráfica de ponteiros (supondo uma máquina de 32 bits)

**1. Ponteiro para carater, não inicializado**

a) Exemplo de declaração em linguagem C:

```
char *p;
```

b) Interpretação gráfica:



c) Ação desenvolvida na tradução para linguagem máquina:

- Definir o registo interno / reservar espaço na memória para alojar um endereço (32 bits)

d) 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.

e) Caso o ponteiro resida na memória, uma possível tradução para *Assembly* do MIPS da sua declaração é:

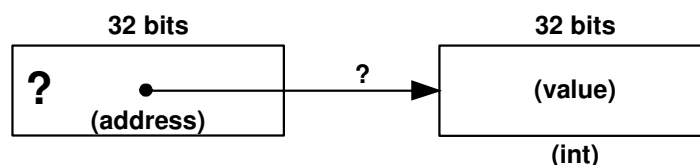
```
p: .space 4      # Reserva 4 bytes de memória
                  # (32 bits) para alojar o
                  # ponteiro. Não há inicialização
```

**2. Ponteiro para inteiro, não inicializado**

a) Exemplo de declaração em linguagem C:

```
int *p;
```

b) Interpretação gráfica:



c) Ação desenvolvida na tradução para linguagem máquina:

- Reservar espaço na memória/registo interno para um endereço (32 bits)

d) Possível tradução para *Assembly* do MIPS (caso o ponteiro resida na memória):

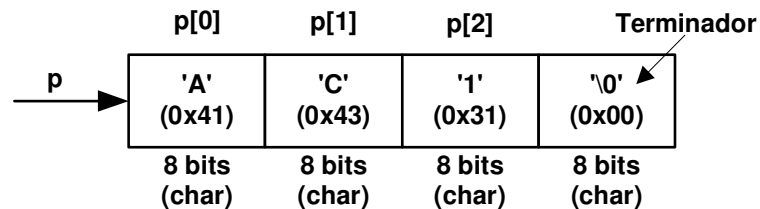
```
p: .space 4      # Reserva 4 bytes de memória
                  # (32 bits) para alojar o
                  # ponteiro. Não há inicialização
```

### 3. Array de caracteres

a) Exemplo de declaração em linguagem C:

```
char p[]="AC1";
```

b) Interpretação gráfica:



c) Ação desenvolvida na tradução para linguagem máquina:

- Reservar espaço na memória para um *array* de caracteres (incluindo para o terminador, o byte 0x00), e efetuar a respetiva inicialização

d) Possível tradução para *Assembly* do MIPS:

```
p:  .asciiz "AC1"    # Reserva 4 bytes de memória e
                    # inicializa-os com os códigos
                    # ASCII dos 3 caracteres e com o
                    # código do terminador (0).
                    # O valor de "p" pode ser obtido
                    # com a instrução "load address"
```

Ou, alternativamente:

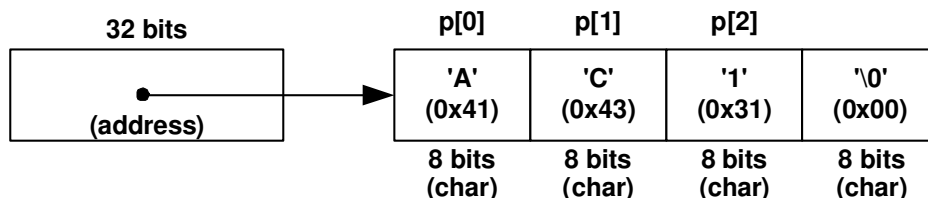
```
p:  .ascii  "AC1"    # Reserva 3 bytes de memória e
                    # inicializa-os com os códigos
                    # ASCII dos 3 caracteres
    .byte  0x00      # Reserva 1 byte e inicializa-o
                    # com o valor 0
```

### 4. Ponteiro para Array de caracteres

a) Exemplo de declaração em linguagem C:

```
char *p = "AC1";
```

b) Interpretação gráfica:



c) Ações desenvolvidas na tradução para linguagem máquina:

- Reservar espaço para um *array* de caracteres e efetuar a respetiva inicialização
- Reservar espaço para um endereço e efetuar a respetiva inicialização



d) Tradução para *Assembly* do MIPS:

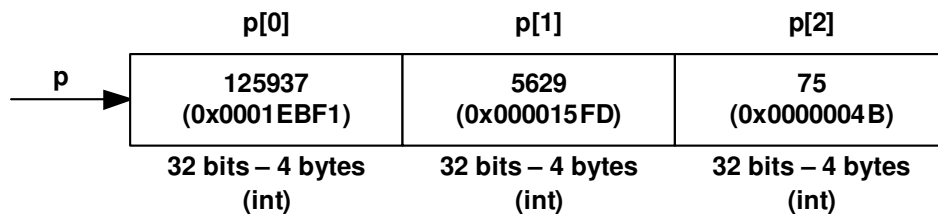
```
str:    .asciiz "AC1"
p:      .word   str # Reserva 4 bytes de memória
                # e inicializa-os com o endereço
                # da primeira posição do array
                # de caracteres (i.e. &array[0]).
                # O valor de "p" pode ser
                # obtido com a instrução "load
                # address"
```

## 5. Array de inteiros

a) Exemplo de declaração em linguagem C:

```
int p[] = {125937, 5629, 75};
```

b) Interpretação gráfica:



c) Ação desenvolvida na tradução para linguagem máquina:

- Reservar espaço para um *array* de inteiros e efetuar a respetiva inicialização

d) Tradução para *Assembly* do MIPS:

```
p:      .word   125937, 5629, 75 #
                # 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 = {125937, 5629, 75};`". Contudo, esta declaração pode ser decomposta em duas, do seguinte modo:

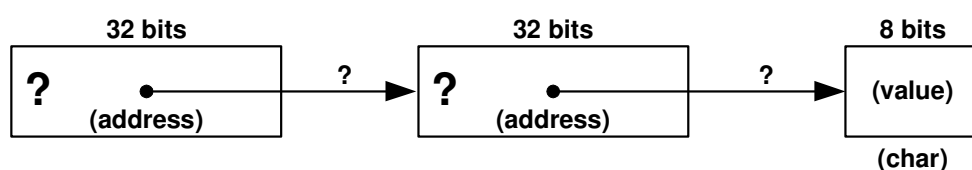
```
int pp[] = {125937, 5629, 75};
int *p = pp;
```

## 6. Ponteiro para ponteiro para carater, não inicializado

a) Exemplo de declaração em linguagem C:

```
char **p;
```

b) Interpretação gráfica:



c) Ação desenvolvida na tradução para linguagem máquina:

- Reservar espaço para um endereço (32 bits)

d) Tradução para *Assembly* do MIPS (caso o ponteiro resida na memória):

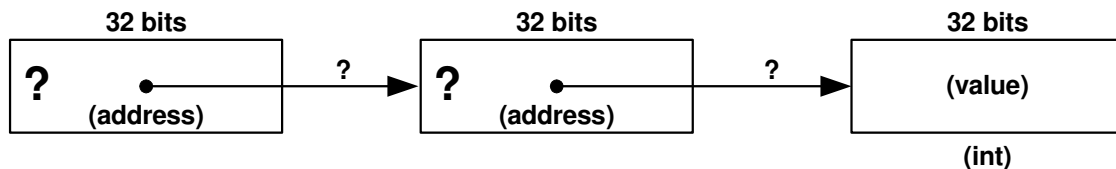
```
p: .space 4      # Reserva 4 bytes na memória para
                  # alojar o ponteiro
```

## 7. Ponteiro para ponteiro para inteiro, não inicializado

a) Exemplo de declaração em linguagem C:

```
int **p;
```

b) Interpretação gráfica:



c) Ação desenvolvida na tradução para linguagem máquina:

- Reservar espaço para um endereço (32 bits)

d) Tradução para *Assembly* do MIPS (caso o ponteiro resida na memória):

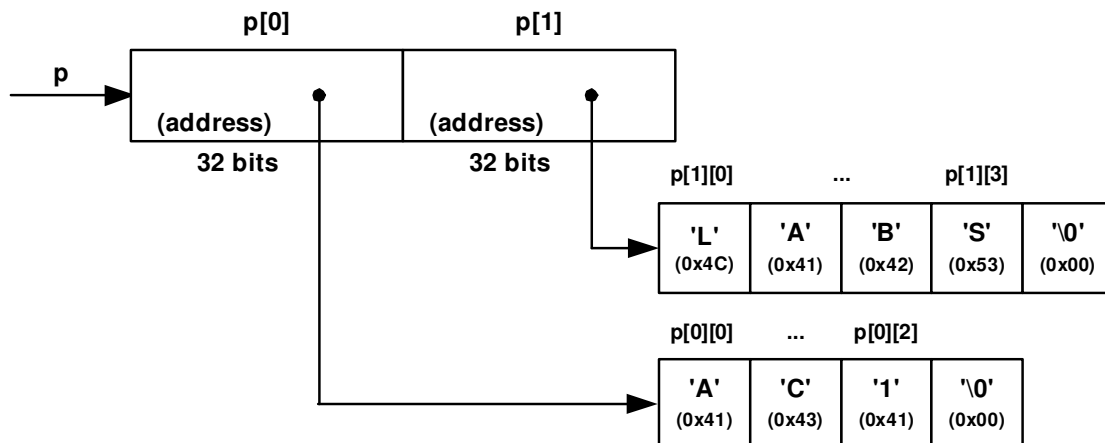
```
p: .space 4      # Reserva 4 bytes na memória para
                  # alojar o ponteiro
```

## 8. Array de ponteiros para carater

a) Exemplo de declaração em linguagem C:

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

b) Interpretação gráfica:



c) Ações desenvolvidas na tradução para linguagem máquina:

- Reservar espaço para os *arrays* de caracteres e efetuar a respetiva inicialização
- Reservar espaço para o *array* de ponteiros (*array* de inteiros) e efetuar a respetiva inicialização

d) Tradução para *Assembly* do MIPS (caso os ponteiros residam na memória):

```
array1: .asciiz "AC1"
array2: .asciiz "LABS"
p:      .word  array1, array2
```