Funções, *arrays* e ponteiros

Programação em Sistemas Computacionais

João Pedro Patriarca (jpatri@cc.isel.ipl.pt, joao.patriarca@isel.pt), Gabinete F.0.23 do edifício F

ISEL, ADEETC, LEIC

Sumário

- Funções
- *Arrays*
- Ponteiros

Sumário

- Funções
- *Arrays*
- Ponteiros

Funções

Bib: (A), cap. 4 {1-4, 8-9}

- Objetivos
  - Separar responsabilidades
  - Organização da solução
  - Abstrair
  - Reutilizar
  - Organizar trabalho em equipa
- Elementos que constituem a assinatura de uma função
  - Nome da função
  - Tipo e nome dos parâmetros
  - Tipo de retorno (`void` se não retornar nada (procedimento))
- Declarada sempre antes de usada
- Irrelevante a localização da definição
- Não é permitida a repetição de nomes

Funções (exemplo)

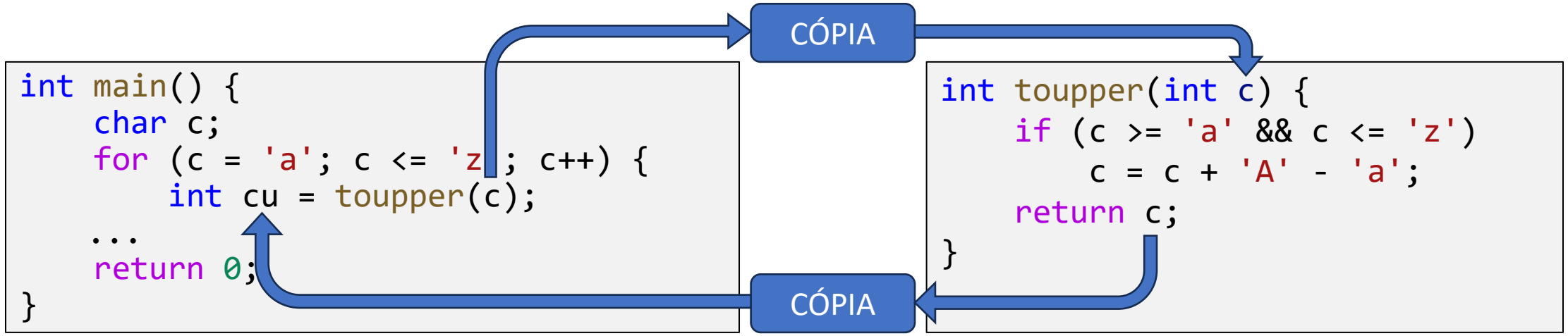
```c
#include <stdio.h>
// Declaração
int tolower(int);
//Declaração com definição
int toupper(int c) {
    if (c >= 'a' && c <= 'z')
        c = c + 'A' - 'a';
    return c;
}
int main() {
    char c;
    for (c = 'a'; c <= 'z'; c++) {
        int cu = toupper(c);
        printf("%c-%d -> upper -> %c-%d\n", c, c, cu, cu);
    }
    for (c = 'A'; c <= 'Z'; c++) {
        int cl = tolower(c);
        printf("%c-%d -> lower -> %c-%d\n", c, c, cl, cl);
    }
    return 0;

}
int tolower(int c) {
    if (c >= 'A' && c <= 'Z')
        c = c + 'a' - 'A' ;
    return c;
}
```

```
a-97 -> upper -> A-65
b-98 -> upper -> B-66
c-99 -> upper -> C-67
...
x-120 -> upper -> X-88
y-121 -> upper -> Y-89
z-122 -> upper -> Z-90
A-65 -> lower -> a-97
B-66 -> lower -> b-98
C-67 -> lower -> c-99
...
X-88 -> lower -> x-120
Y-89 -> lower -> y-121
Z-90 -> lower -> z-122
```

Chamada e retorno de uma função

Bib: (A), cap. 4 {1-4, 8-9}

- Chamada
  - Os argumentos (parâmetros atuais) são copiados para os parâmetros da função (parâmetros formais) -> chama-se passagem por cópia
  - Correspondência por ordem
- Retorno
  - O resultado da expressão do retorno é copiado para o contexto da chamada



Variáveis globais, locais e parâmetros

Bib: (A), cap. 4 {1-4, 8-9}

- Variáveis globais
  - Definidas fora de qualquer função
  - Alocação estática
  - Tempo de vida do programa
  - Iniciadas explicitamente ou implicitamente (com 0)
  - Visíveis em todo o programa
- Variáveis locais e parâmetros
  - Definidas dentro de uma função
  - Alocação automática
  - Tempo de vida da função
  - Não são iniciadas implicitamente
  - Visíveis apenas dentro da função

```
int var1, var2 = 100;
int inc_vars(int v) {
    var1++; var2++;
    return ++v;
}
int main() {
    int var3 = 10;
    for (int i = 0; i < 10; i++)
        var3 = inc_vars(var3);
    var1 += 1; var2 += 1;
    printf("var1 = %d, var2 = %d,"
           " var3 = %d\n",
           var1, var2, var3);
    return 0;
}
```

```
var1 = 11, var2 = 111, var3 = 20
```

Sumário

- Funções
- *Arrays*
- Ponteiros

Problema

- Histograma com o número de dígitos presentes num ficheiro de texto
- Exemplo:

```
XXX:~/test$ ./hist_digits < test_file.txt
0: *******
1:
2: ***
3: *****
4:
5:
6: *
7: ****
8: *********
9: **
```

- Dificuldade: como manter em memória o número de ocorrências de cada dígito?

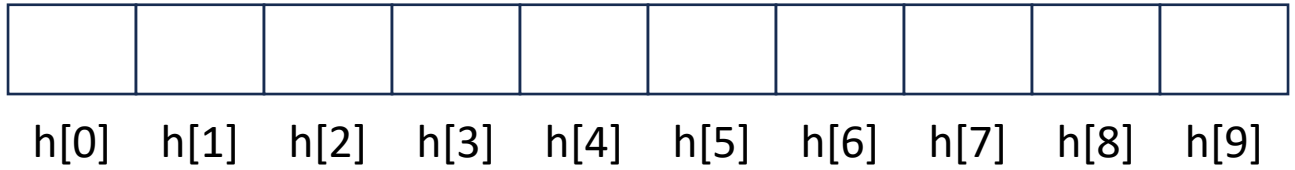
Arrays

Criação e inicialização

Bib: (A), cap. 5.3

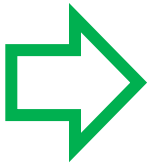
- Definição: agregado de elementos em memória do mesmo tipo
- Dimensão fixa e constante
  - A dimensão é determina na declaração do *array*
  - Não existe qualquer campo com a dimensão (*h.Length*)

```
int h[10];
```



- Obter a capacidade de um *array*

```
int cap = sizeof(h)/sizeof(h[0]);
printf("cap=%d\n", cap);
```



```
cap=10
```

- Permite inicialização explícita no momento da declaração

```
int a[4] = {0, 1, 0, 1};// Declaração e inicialização
int b[] = {0, 1, 0};    // Capacidade implícita (3 inteiros)
int c[20] = {-1};       // Primeiro elemento com -1 e restantes com 0
```

Arrays

Acesso

Bib: (A), cap. 5.3

- Aceder a elementos do *array*: por indexação através dos símbolos []
  - O primeiro elemento é acedido sempre pelo índice 0
  - O último elemento é acedido sempre pela capacidade do *array* -1

```
int a[4] = {0, 1, 0, 1};
int first = a[0];
int last = a[3];
int overlast = a[4]; // Não deteta acesso fora dos limites
a[1] = 2;
```

- Não existe afetação de *arrays*

```
int x[5], y [5];
x = y; // Erro de compilação
```

Problema: histograma de dígitos

```
void count_digits(int h[]) {
    char c;
    while (c=getchar(), c != EOF)
        if (c >= '0' && c <= '9')
            h[c-'0'] += 1;
}
void print_digits_hist(int hist[]) {
    for (int i = 0; i < 10; i++) {
        printf("%d: ", i);
        for (int j = 0; j < hist[i]; j++)
            printf("*");
        printf("\n");
    }
}
int main() {
    int hist[10] = {0};
    count_digits(hist);
    print_digits_hist(hist);
    return 0;
}
```

- Como é passado a uma função o argumento do tipo *array*?
  - É passado por cópia o endereço do primeiro elemento

Exercício

1. Ordenar um *array* de inteiros (*Long*) de acordo com o algoritmo *selection sort*

```
void sort( long a[], int size );
void swap( long a[], int i1, int i2 );
void print_array( long a[], int size );
```

Sumário

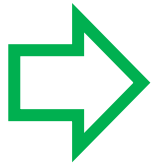
- Funções
- *Arrays*
- Ponteiros

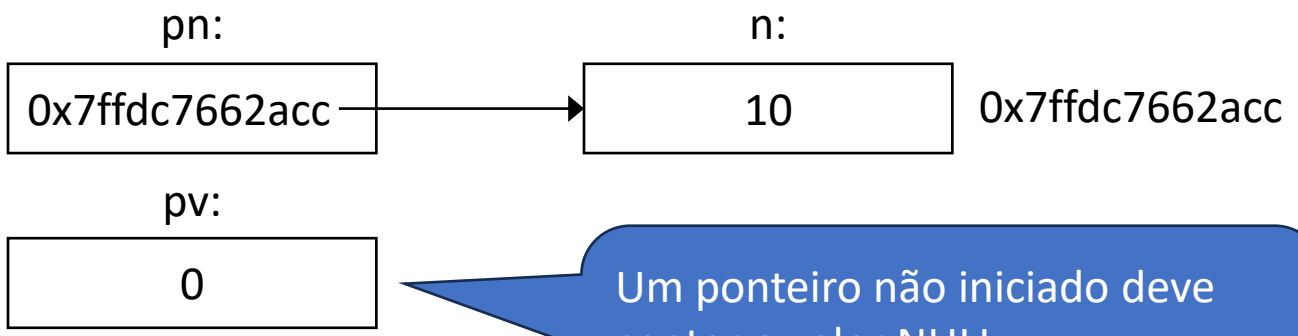
Ponteiros

Bib: (A), cap. 5 {.1-.5}

- Um ponteiro guarda uma posição de memória / endereço de memória
- Declaração e iniciação

```
int n = 10;
int * pn, *pv;
pn = &n;
pv = NULL;
```





- Operador *i* comercial (&)
  - O operador & significa “endereço de”
  - Aplicável a objetos em memória
- Operador asterisco (*)
  - O operador * significa “desreferência” ou “indireção”
  - Aplicado a um ponteiro acede-se ao objeto em memória

```
printf("pn=%p, *pn=%d, pv=%p\n",
       (void*)pn, *pn, (void*)pv);
```

```
pn=0x7ffe3bf2ab84, *pn=10, pv=(nil)
```

Ponteiros

Exemplo

```c
int i = 10, y1, y2;
int *ip1 = &i, *ip2;
ip1 = *ip1 + 10; // ⇔ *ip1+=10
y1 = ++*ip1;      // Desreferencia e incrementa valor
ip2 = ip1;
y2 = *++ip2;      // Incrementa ponteiro e desreferencia
printf("ip1=%p, ip2=%p, y1=%d, y2=%d\n", (void*)ip1, (void*)ip2, y1, y2);
```

```
ip1=0x7fffdd5a538c, ip2=0x7fffdd5a5390, y1=21, y2=21
```

Ponteiros e *const*

- Ponteiro para *char* não alterável (*const char **)
- Ponteiro não alterável para *char* (*char * const*)

```
char c1 = 1, c2 = 2;
const char * p1 = &c1;
char * const p2 = &c2;
// *p1 = 10; // Erro de compilação
*p2 = 20; p1 = &c2;
// p2 = &c1; // Erro de compilação
printf("p1=%p, *p1=%d, p2=%p, *p2=%d\n", p1, *p1, p2, *p2);
printf("c1=%d, c2=%d\n", c1, c2);
```

```
p1=0x7fff0f3f0a27, *p1=20, p2=0x7fff0f3f0a27, *p2=20
c1=1, c2=20
```

Ponteiros e *arrays*

Bib: (A), cap. 5 {.1-.5}

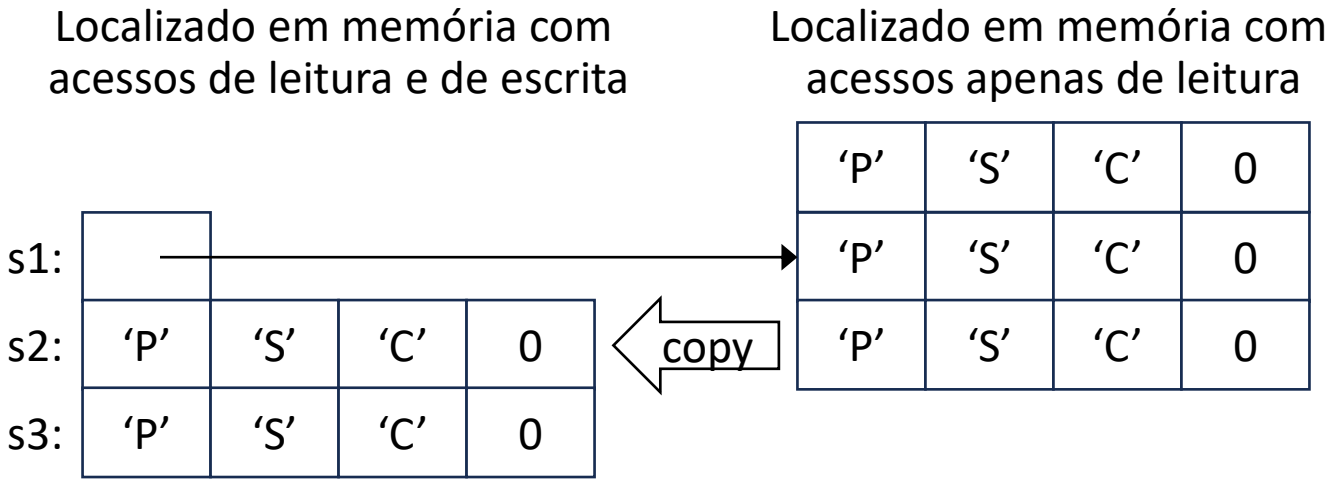
- O nome de um *array* corresponde a um ponteiro inalterável para o primeiro elemento em memória

```
char a[5] = {'1', '2', '3'};
char *pa;
pa = a;   // ⇔ pa = &a[0];
//a = pa; // Erro de compilação
//a++;    // Erro de compilação
printf("a=%p, pa=%p, a[0]=\'%c\', *pa=\'%c\'\n", a, pa, a[0], *pa);
// a[i] ⇔ *(pa+i)
// &a[i] ⇔ pa+i
printf("a[2]=%d, *(pa+2)=%d, &a[4]=%p, pa+4=%p\n", a[2], *(pa+2), &a[4], pa+4);
```

```
a=0x7ffe0587b503, pa=0x7ffe0587b503, a[0]='1', *pa='1'
a[2]=51, *(pa+2)=51, &a[4]=0x7ffe0587b507, pa+4=0x7ffe0587b507
```

Ponteiros e *strings*

- Uma *string* é uma sequência de caracteres terminados pelo valor 0
- A expressão *"PSC"*
  - Reserva memória estática para um *array* de 4 caracteres
  - O *array* tem os códigos dos caracteres e um zero
  - O seu valor é o endereço para o primeiro carácter
  - O tipo é `const char *`




```c
const char *s1 = "PSC";           // Sequência inalterável
char s2[] = "PSC";                // Sequência alterável
char s3[5] = {'P', 'S', 'C', 0}; // Sequência alterável
printf("s1=%s\ns2=%s\ns3=%s\n", s1, s2, s3);
printf("cap(s2) = %d bytes\n",
       (int)(sizeof(s2)/sizeof(s2[0])));
```

```
s1=PSC
s2=PSC
s3=PSC
cap(s2) = 4 bytes
```

Ponteiros e *strings*

Exemplo: *void strcpy(char * dst, const char * src);*

Bib: (A), cap. 5.5

```c
void strcpy_v1(char * dst,
               const char * src) {
    int i = 0;
    while ((dst[i] = src[i]) != '\0')
        i++;
}
```

```c
void strcpy_v3(char * dst,
               const char * src) {
    while ((*dst++ = *src++) != '\0')
        ;
}
```

```c
void strcpy_v2(char * dst,
               const char * src) {
    while ((*dst = *src) != '\0') {
        dst++;
        src++;
    }
}
```

```c
void strcpy_v4(char * dst,
               const char * src) {
    while ((*dst++ = *src++))
        ;
}
```

- Outras funções da biblioteca C que manipulam *strings*:

```c
int strcmp( const char *ls, const char *rs );// Compara duas strings
size_t strlen( const char *str );            // Devolve o número de caracteres
char *strcat( char *dest, const char *src ); // Concatena string src à string dest
```

Aritmética de ponteiros (em construção)

- Considerando dois ponteiros do tipo T

```
T * p1, * p2;
```

- Soma com valor inteiro N
  - Posição N*`sizeof(T)` bytes à frente

```
++pi1;
pi2 = pi1 + 10;
pi1 += 2;
```

- Subtracção com valor inteiro N
  - Posição N*`sizeof(T)` bytes a trás

```
--pi1;
pi2 = pi1 - 10;
pi1 -= 2;
```

- Diferença entre ponteiros
  - Distância em elementos de `sizeof(T)` bytes

```
int dif;
dif = pi2 - pi1;
```

- Não é permitido:
  - Adicionar, multiplicar, dividir dois ponteiros
  - Mascarar, deslocar um ponteiro
  - Adicionar com um `float` ou `double`

Exercícios

1. Ordenar um *array* de inteiros (`unsigned short`) de acordo com o algoritmo *selection sort*

```
void sort( short * a, int size );
void swap( short * a, int i1, int i2 );
void print_array( short * a, int size );
```

2. Descompactar uma data recorrendo a função `getbits`

```
void date_unpack( short date, int * py, int * pm, int * pd );
int getbits( short v, int high, int low );
```

3. Comparar duas *strings* (`ls` e `rs`)
   - Retorna < 0 se `ls` < `rs`, = 0 se `ls`==`rs` e > 0 se `ls` > `rs`

```
int strcmp( const char * ls, const char * rs );
```

4. Concatenar a *string* `src` na *string* `dst`

```
char * strcat( char * dst, const char * src );
```