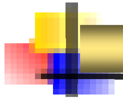
Ponteiros



Prof. Dr. Silvio do Lago Pereira

Departamento de Tecnologia da Informação

Faculdade de Tecnologia de São Paulo

Ponteiros

- **Ponteiro é uma referência a uma posição de memória.**
- **Um ponteiro contém o endereço de outra variável.**
- **Se um ponteiro p guarda o endereço de uma variável v, dizemos que p aponta v.**

Importante

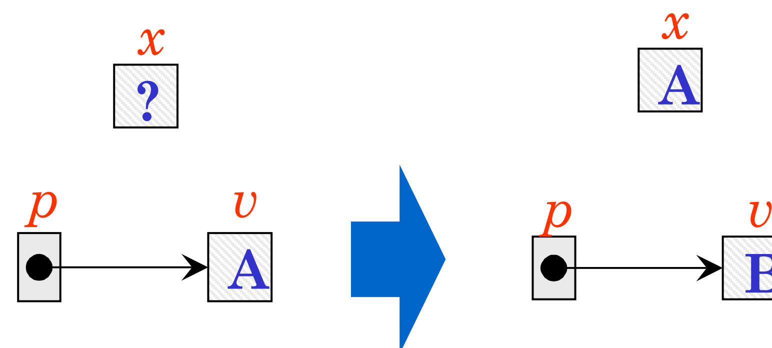
- O tipo de um ponteiro depende do tipo da variável que ele aponta.
- Para indicar que uma variável é um ponteiro, devemos prefixar seu nome com um * na sua declaração.

```
char *p;

char v = 'A';

p = &v;
```

Importante

- Para acessar o local de memória apontado usamos um ∗ prefixando o nome da variável.
- Se p é um ponteiro, ∗p representa o conteúdo da área de memória apontada por p.
- Se p contém o endereço de v, então alterar ∗p equivale a alterar v.

```
char *p, v='A', x;
p = &v;
x = *p;
*p = 'B';
```



Passagem por valor/referência

- **C passa argumentos por valor.**
- **quando uma função é chamada, um novo espaço de memória é alocado para cada um de seus parâmetros.**
- **nenhuma alteração feita pela função pode afetar os argumentos originais.**
- **funções ficam impedidas de acessar variáveis declaradas noutras funções.**

Necessidade de referência

```
#include <stdio.h>
void perm(int p, int q) {
   int x;
   x = p;
   p = q;
   q = x;
}
void main(void) {
   int a=3, b=7;
   perm(a,b);
   printf("%d %d", a, b);
}
```

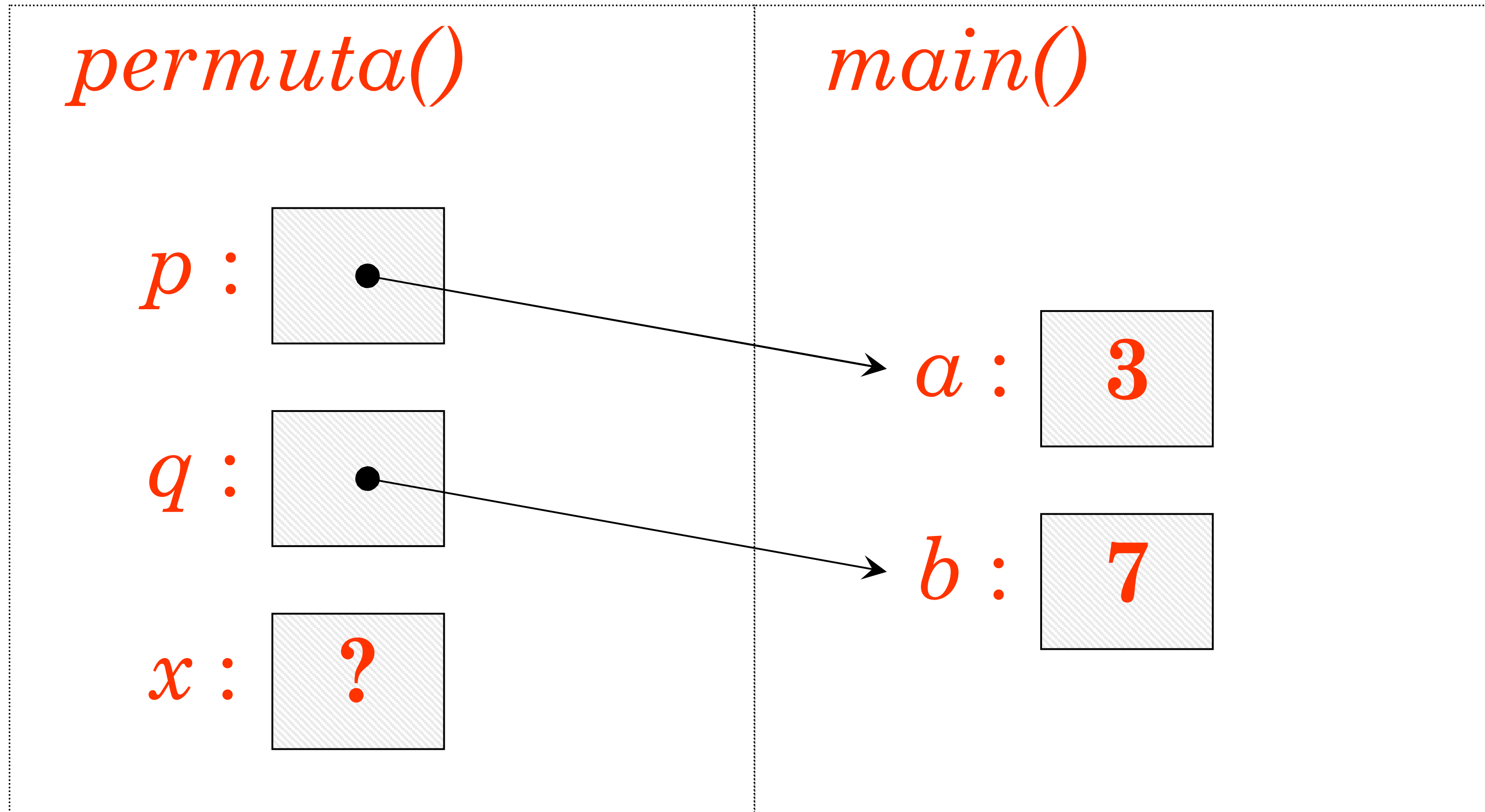
Alocação de parâmetros

$perm()$		$main()$	
p :	**3**	a :	**3**
q :	**7**	b :	**7**
x :	**?**		

Usando referência

```c
#include <stdio.h>
void permuta(int *p, int *q) {
    int x;
    x = *p;
    *p = *q;
    *q = x;
}
void main(void) {
    int a=3, b=7;
    permuta(&a,&b);
    printf("%d %d", a, b);
}
```

Alocação de parâmetros



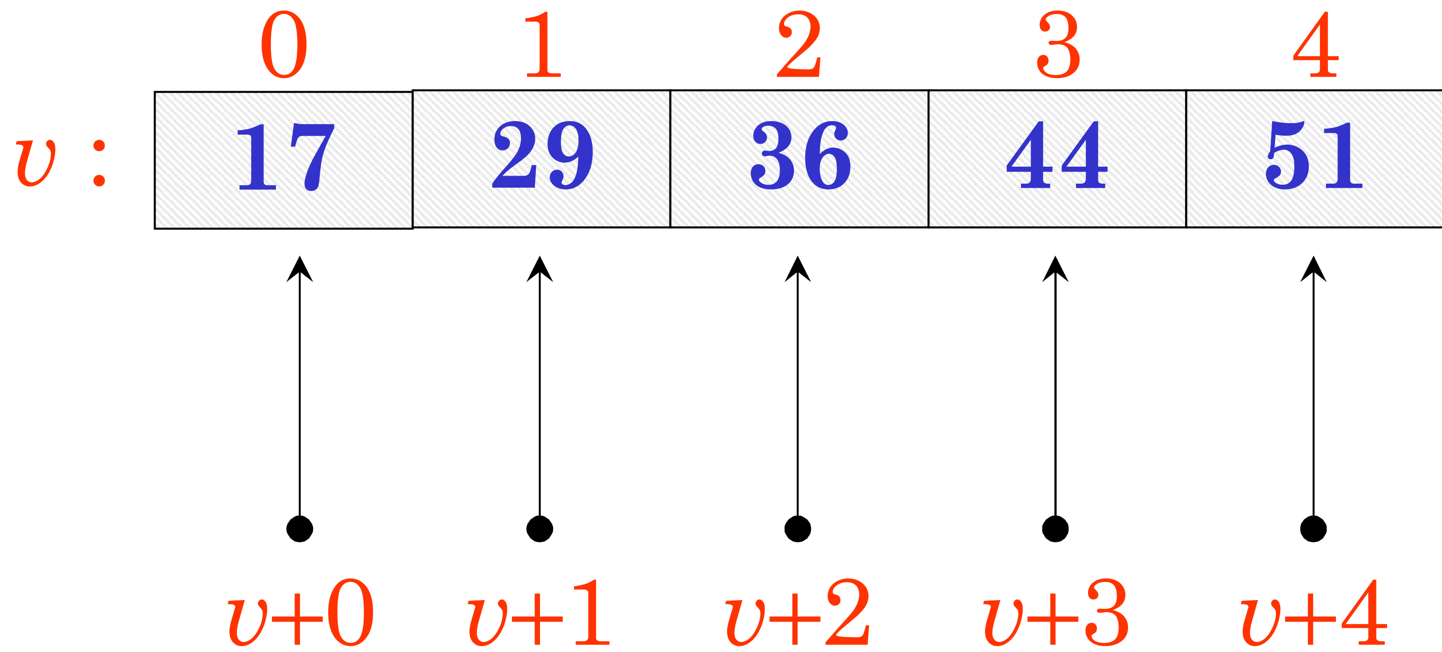
Aritmética de ponteiros

- Uma operação freqüente é a adição de ponteiros a números inteiros.
- Essa operação é implicitamente usada quando trabalhamos com vetores.
- Quando escrevemos $v[i]$ o compilador automaticamente executa a adição v+i.
- v+i é o endereço do item que está a i posições do início do vetor v.
- Então, $v[i]$ equivale a $*(v+i)$.

Usando ponteiros para vetores

```
#include <stdio.h>

void main(void) {
   static int v[5] = {17,29,36,44,51};
   int i;

   for(i=0; i<5; i++)
      printf("%d ", *(v+i) );
}
```

Representação

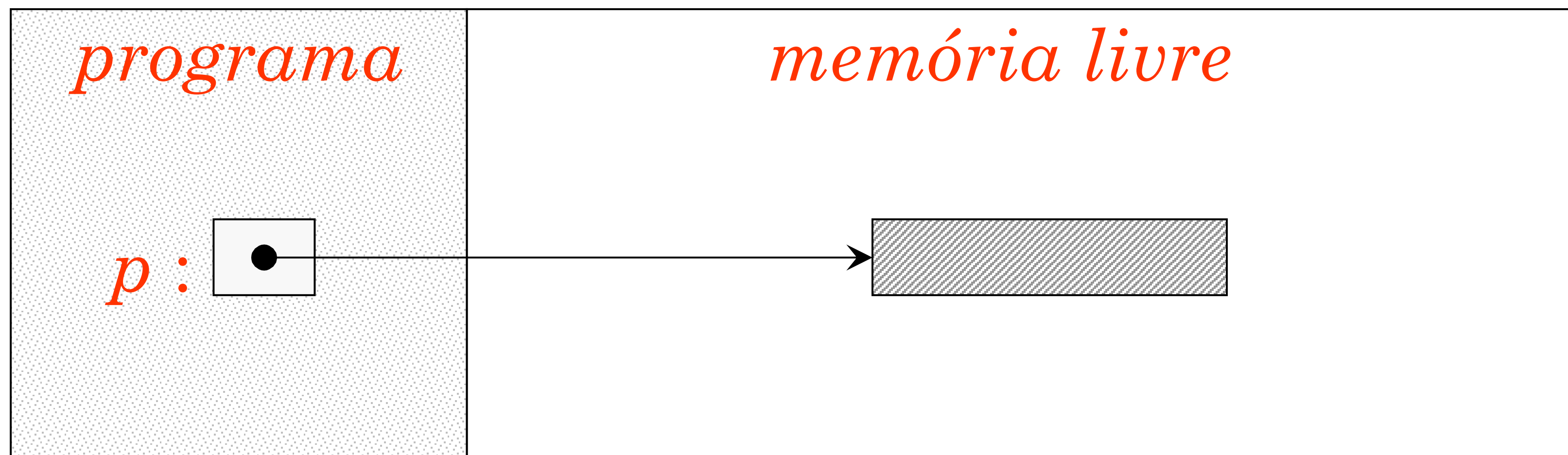


Importante

- **O resultado da adição depende do tipo do ponteiro;**
- **Alguns tipos requerem mais memória do que outros.**
- **Quando incrementado, um ponteiro deve saltar para o próximo item na memória e não para o próximo byte.**

Alocação dinâmica de memória

- **Ao carregar um programa, parte a memória fica livre;**
- **Podemos usar malloc() para alocar parte dessa área;**
- **O acesso à área alocada é feito através de ponteiro;**
- **A função malloc() está definida em alloc.h;**



Malloc( )

p = malloc(n);

- Aloca uma área de n bytes;
- sizeof determina a quantidade a ser alocada;
- Devolve o endereço da área alocada;
- Armazena esse endereço no ponteiro p;
- Devolve NULL se não existe espaço suficiente para efetuar a alocação.

Vetores dinâmicos

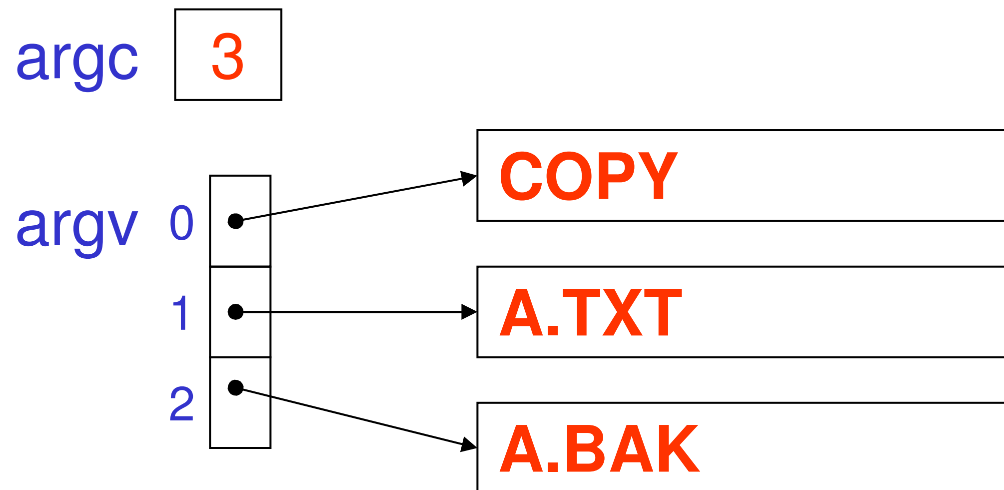
```
void main(void) {
   int *v, n, i;
   printf("\nTamanho do vetor? ");
   scanf("%d",&n);
   v = malloc( n * sizeof(int) );
   if( v == NULL ) exit(1);
   for(i=0; i<n; i++) {
      printf("\n%do. Valor? ");
      scanf("%d",&v[i]);
   }
   while(i>0 ) printf("%d ",v[--i]);
}
```

Importante

- **Ao final da execução, todas as áreas alocadas com malloc() são liberadas automaticamente;**
- **Para liberar explicitamente uma dessas áreas usamos a função free().**
- **Essa função exige como argumento o endereço da área a ser desalocada.**

Argumentos da linha de comando

```
C:\TC>COPY A.TXT A.BAK
```




```
int argc = 3;

char *argv[] = {"COPY","A.TXT","A.BAK"};
```

Programa eco

```
Z:\> eco um dois tres <enter>
um
dois
tres
Z:\> _
```

```
#include <stdio.h>
void main(int argc, char *argv[]) {
   int i;
   for(i=1; i<argc; i++)
      puts(argv[i]);
}
```

Programa hist

```
Z:\> hist 3 5 2 <enter>
***
*****
**
Z:\> _
```

```
#include <stdio.h>
#include <stdlib.h>
void main(int argc, char *argv[]) {
   int i, j;
   for(i=1; i<argc; i++) {
      putchar('\n');
      for(j=0; j<atoi(argv[i])); j++)
         putchar('*');
   }
}
```

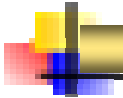
Programa calc

```
#include <stdio.h>
#include <stdlib.h>
void main(int argc, char *argv[]) {
   float a, b, c;
   if( argc!=3 ) {
      puts(“Uso: calc <val> <op> <val>”);
      exit(1);
   }
   a = atof(argv[1]);
   b = atof(argv[3]);
```

Programa calc

```
    switch( argv[2][0] )
        case '+': c = a+b; break;
        case '-': c = a-b; break;
        case '*': c = a*b; break;
        case '/': c = a/b; break;
        default: puts("erro"); exit(2);
    }
    printf("\nRes: %.1f", c);
}
```

Operadores bit-a-bit



Prof. Dr. Silvio Lago

Departamento de Tecnologia da Informação

Faculdade de Tecnologia de São Paulo

Notação Hexadecimal

- **Para facilitar o raciocínio com operadores bit-a-bit é melhor usar a notação hexadecimal (0xHH)**
- **Cada dígito hexa corresponde a 4 bits**

pesos: $2^3\ 2^2\ 2^1\ 2^0$

$\times\ \times\ \times\ \times$

binário: $1\ 0\ 1\ 0 \Rightarrow 1\times 8 + 0\times 4 + 1\times 2 + 0\times 1 = 10_D$

dec/hex: $10_D = A_H$

- **Exemplo**

0x3A7C = 0011 1010 0111 1100

Operadores

- **Não: ~**
  - **~ 0x51 = ~(0101 0001)**

    = (1010 1110)

    = 0xAE
- **E: &**
  - **0x62 & 0xAB = (0110 0010)**

    & (1010 1011)

    = (0010 0010) = 0x22

Operadores

- **Ou: |**
  - **0x62 | 0xAB = (0110 0010)**
    | (1010 1011)
    = (1110 1011) = 0xED

- **Ou-exclusivo: ^**
  - **0x62 ^ 0xAB = (0110 0010)**
    ^ (1010 1011)
    = (1100 1001) = 0xC9

Operadores

- **Deslocamento à esquerda: <<**
  - **0x62 << 2 = (0110 0010) << 2**
    = (1000 1000) = 0x88

- **Deslocamento à direita: >>**
  - **0x62 >> 3 = (0110 0010) >> 3**
    = (0000 1100) = 0x0C
  - **Usar unsigned para garantir preenchimento com zeros!**

Ligar um grupo de bits

- Ligar os bits 0,1 e 6 de um byte

valor do byte = 1010 0001

|

máscara = 0100 0011

Resultado = 1110 0011

Desligar um grupo de bits

- Desligar os bits 0, 4 e 7 de um byte

valor do byte = 1010 1101

&

máscara = 0110 1110

Resultado = 0010 1100

Verificar se um bit está ligado

- Verificar se o bit 3 de um byte está ligado

valor do byte = 1010 1101

&

máscara = 0000 1000

Resultado = 0000 1000

Alternar um grupo de bits

- Alternar os bits 0, 4 e 7 de um byte

valor do byte = 1010 1101

^

máscara = 1001 0001

Resultado = 0011 1100

Isolar o valor de um grupo de bits

- Determinar o valor dos bits 2 a 5 de um byte

valor do byte = 1010 1101

<< 2

= 1011 01**00**

>> 4

Resultado = **0000** 1011

Status da impressora

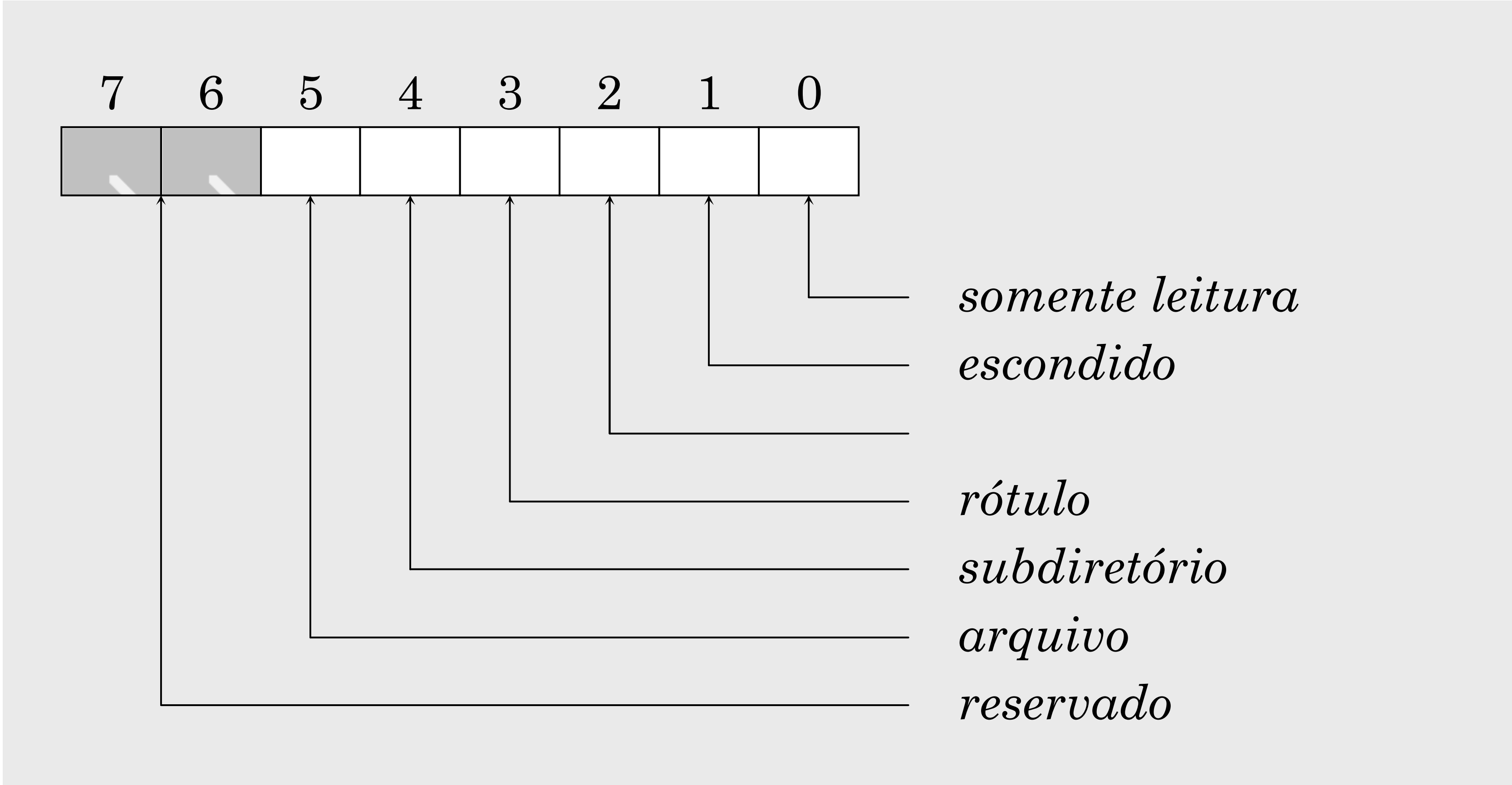
Obtendo o status da impressora

- int biosprint(int cmd, int byte, int port);
  - cmd: 0=enviar, 1=iniciar, 2=obter status
  - byte: a valor a ser enviado quando cmd=0
  - port: 0=LPT1, 1:LPT2, ...
  - Devolve byte de estatus da impressora
  - Necessita bios.h

Impressora tem papel?

```
#include <stdio.h>
#include <bios.h>
#define LPT1 0
void main(void) {
   int status = biosprint(2,0,LPT1);
   if( status & 0x20 )
       puts("Impressora sem papel");
   else
       puts("Impressora com papel");

}
```

Atributo de arquivo



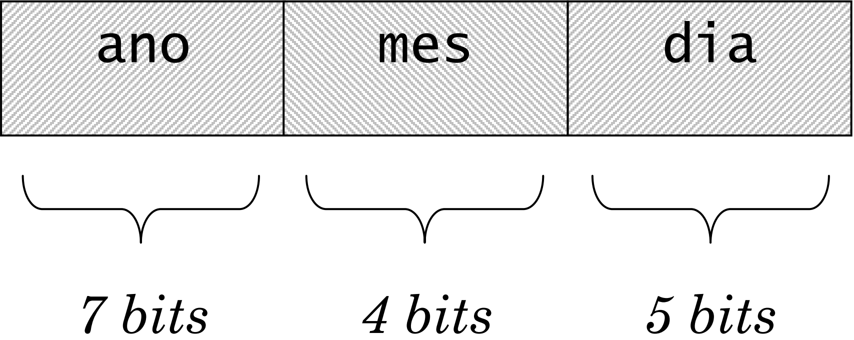
Obtendo atributo de arquivo

- int _chmod(char *name, int cmd [, int byte]);
  - name: nome do arquivo
  - cmd: 0= lê atributo 1= grava atributo
  - byte: novo valor do atributo
  - devolve atributo ou -1 se arquivo não encontrado
  - necessita io.h

Escondendo um arquivo...

```
#include <stdio.h>
#include <io.h>
void main(void) {
   char arq[200];
   int atrib;
   printf("\nArquivo? ");
   gets(arq);
   atrib = _chmod(arq,0);
   if( atrib != -1 ) {
       atrib = atrib ^ 0x02;
       _chmod(arq,1,atrib);
   }
   else puts("Arquivo não encontrado");
}
```

Datas compactadas



Busca em diretórios

- Dados sobre arquivos (dir.h), mantidos pelo SO.

```
struct ffblk {
    char        ff_reserved[21];
    char        ff_attrib;
    unsigned    ff_ftime;
    unsigned    ff_fdate;
    long        ff_fsize;
    char        ff_name[13];
};
```

Obtendo dados sobre arquivos

- int findfirst(char *name, struct ffblk *fd, int attrib);
- int findnext(struct ffblk *fd);
  - name: nome do arquivo pode conter * e ?
  - fd: dados sobre o arquivo encontrado
  - attrib: atributo do arquivo procurado (todos=0xFF)
  - devolve 0 se sucede na busca
  - necessita dir.h

Listando arquivos...

```
#include <stdio.h>
#include <dir.h>

void lista(char *path) {
  struct ffblk fd;
  int ok = !findfirst(path,&fd,0xFF);

  while( ok ) {
    printf("\n%s", fd.ff_name);
    ok = !findnext(&fd);
  }
}
```

Listando arquivos...

```
void main(int argc, char *argv[]) {
  char path[200];

  if( argc!=2 ) {
     printf("Uso: lista <caminho>\n");
     exit(0);
  }

  lista(argv[1]);
}
```

Scanning...

```
#include <stdio.h>
#include <dir.h>
#include <dos.h>

void scan(char path[]) {
   struct ffblk f;
   int n, ok;

   n = strlen(path);
   strcpy(path+n,"\\*.*");
   ok = !findfirst(path,&f,0xFF);
   ...
```

Scanning...

```
...
   while( ok ) {
      strcpy(path+n+1,f.ff_name);
      if( f.ff_attrib&FA_DIREC &&
          f.ff_name[0]!='.' )
          scan(path);
      else
          printf("\n%s",path);
      ok = !findnext(&f);
   }
}
```

Scanning...

```
…

void main(void) {
   static char path[1000] = "C:";
   int x, y;

   clrscr();
   scan(path);
   getch();
}
```

Fim