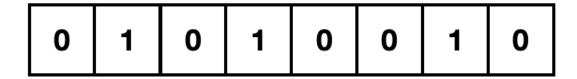
Programação Estruturada

Aula 14 - Ponteiros

Yuri Malheiros (yuri@ci.ufpb.br)

- O primeiro passo para entendermos ponteiros é visualizar o que eles representam a nível de máquina
- A memória dos computadores são dividias em bytes e cada byte guarda 8 bits

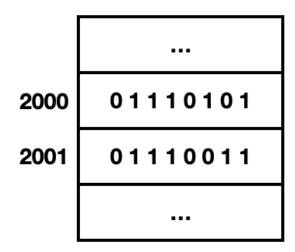


• Cada byte tem um endereço único para diferenciá-lo dos outros bytes na memória

• Se uma memória tem n bytes, então os endereços vão de 0 até n-1

0	01010011
1	01110101
2	01110011
3	01010011
4	01101110
n-1	01000011

- Cada variável de um programa ocupa um ou mais bytes na memória
- O endereço do primeiro byte é o endereço da variável
- Se uma variável i ocupa dois bytes, então ela ser armazenada assim:



- Endereços de variáveis podem ser armazenados em ponteiros
- Quando guardamos um endereço de uma variável i num ponteiro p dizemos que
 - p aponta para i

- Um ponteiro nada mais é do que uma variável que guarda um endereço
- Para indicar que p aponta para i usamos:



Declarando ponteiros

- Um ponteiro é declarado como uma variável
- A única diferença é que o nome do ponteiro deve começar com um asterisco
 - ∘ int *p
- Esta declaração indica que p é um ponteiro capaz de apontar para valores do tipo int
- Para outros tipos, temos:
 - o double *q
 - ∘ char *r

- C fornece operadores que são usados especificamente com ponteiros
- Para encontrar um endereço usamos o operador de endereço &
 - Se x é uma variável, então &x é o endereço dela na memória

Operador de indireção

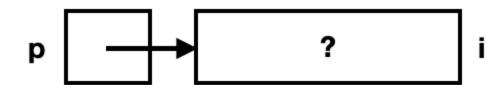
- Para acessar o valor apontado por um ponteiro usamos o operador de indireção *
 - Se p é um ponteiro, então *p representa o valor que p aponta

• Declarar um ponteiro reserva um espaço na memória, mas não aponta para um valor

```
int *p /* não está apontando para um valor */
```

- É crucial inicializar p antes de usá-lo
- Uma forma de inicializar um ponteiro é atribuir o valor do endereço de uma variável

```
int i, *p;
p = &i;
```



• Também podemos inicializar o ponteiro na sua declaração

```
int i;
int *p = &i;
```

Operador de indireção

- Dado que um ponteiro foi inicializado, nós podemos usar o operador de indireção *
 para acessar o conteúdo apontado
- Se p aponta para i, podemos imprimir o valor de usando printf("%d\n", *p);

Operador de indireção

Enquanto p apontar para i , *p é um "apelido" para i
 *p tem o mesmo valor de i
 Mudar um valor de *p muda o valor de i

```
int i, *p;
p = &i;
i = 1;

printf("%d\n", i);
printf("%d\n", *p);

*p = 2;

printf("%d\n", i);
printf("%d\n", *p);
```

Atribuição de ponteiros

- C permite o uso do operador de atribuição para copiar ponteiros
- Suponha: int i, j, *p, *q
- O comando p = &i é uma atribuição de ponteiro
- O comando q = p também é
 - o p e q estão apontando para i
 - Atribuir valores para *p ou *q também altera i

Atribuição de ponteiros

- Cuidado para não confundir q = p com *q = *p
- No primeiro, q vai apontar para o mesmo endereço que p aponta
- No segundo, estamos substituindo o valor que p aponta (no caso, o valor i) pelo valor que q aponta

- Vimos anteriormente que um valor passado para uma função é copiado
 - Uma função não pode modificar uma variável passada na sua chamada
- Ponteiros oferecem uma solução para ese problema
- Ao invés de passar a variável x como argumento, passamos um ponteiro &x

• Precisamos declarar o parâmetro como um ponteiro

```
void decompor(double x, long *int_part, double *frac_part) {
  *int_part = (long) x;
  *frac_part = x - *int_part;
}
```

• Para chamar a função: decompor(3.14159, &x, &y);

- É isso que o scanf faz
 - o scanf("%d", &x);
- Ao passar &x passamos um ponteiro para x
- O scanf vai conseguir escrever um valor na variável x

- Sempre devemos usar & no scanf?
 - Agora sabemos que não
 - Nós temos que passar um ponteiro

```
int i, *p;

p = &i;
scanf("%d", p);

printf("i=%d\n", i);
printf("*p=%d\n", *p);
```

Ponteiros como retorno

- Uma função pode retornar um ponteiro
- A função abaixo recebe dois ponteiros para inteiros e retorna um ponteiro para o maior inteiro

```
int *max(int *a, int *b) {
   if (*a > *b)
     return a;
   else
     return b;
}
```

• Veremos exemplos mais práticos desse caso em aulas futuras