

Introdução aos ponteiros

Graziela Santos de Araújo

Faculdade de Computação
Universidade Federal de Mato Grosso do Sul

Algoritmos e Programação II

Conteúdo da aula

- 1 Introdução
- 2 Variáveis ponteiros
- 3 Operadores de endereçamento e de indireção
- 4 Ponteiros em expressões
- 5 Exercícios

- ▶ Ponteiros ou apontadores (do inglês *pointers*)
- ▶ Característica da linguagem C (mais poder e flexibilidade)
- ▶ Estruturas de dados complexas, modificação de argumentos passados a funções, alocação dinâmica de memória, etc.

Variáveis ponteiros

- ▶ **Indireção**, isto é, **acesso indireto** a um valor armazenado em algum ponto da memória
- ▶ **Ponteiro** é uma variável que armazena um valor especial, que é um endereço de memória, e por isso nos permite acessar indiretamente o valor armazenado nesse endereço
- ▶ A memória de um computador é constituída de muitas posições dispostas continuamente, cada qual podendo armazenar um valor na base binária
- ▶ Ou seja, a memória é um grande vetor que pode armazenar valores na base binária e que, por sua vez, esses valores podem ser interpretados como valores de diversos tipos
- ▶ Os índices desse vetor, numerados seqüencialmente a partir de 0 (zero), são chamados de **endereços de memória**

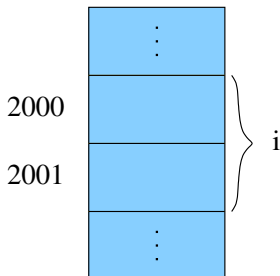
Variáveis ponteiros

0	00010011
1	11010101
2	00111000
3	10010010
	.
	.
	.
n-1	00001111

Variáveis ponteiros

`int i;`

*i é o que pra nós?
e para o computador?*



Variáveis ponteiros

- ▶ Quando armazenamos o endereço de uma variável i em uma variável ponteiro p , dizemos que p **aponta para** i
- ▶ Um ponteiro nada mais é que um endereço e uma variável ponteiro é uma variável que pode armazenar endereços
- ▶ Ao invés de mostrar endereços como números, usaremos uma notação simplificada para indicar que uma variável ponteiro p armazena o endereço de uma variável i : mostraremos o conteúdo de p – um endereço – como uma flecha orientada na direção de i

Variáveis ponteiros



Variáveis ponteiros

- ▶ Declaração de uma variável ponteiro:

```
int *p;
```

- ▶ A linguagem C obriga que toda variável ponteiro aponte apenas para objetos de um tipo particular, chamado de **tipo referenciado**
- ▶ variável ponteiro = ponteiro

Operadores de endereçamento e de indireção

- ▶ Para obter o endereço de uma variável, usamos o **operador de endereçamento (ou de endereço)**, cujo símbolo é **&**
- ▶ Se v é uma variável, então $\&v$ é seu endereço na memória
- ▶ Para ter acesso ao objeto que um ponteiro aponta, temos de usar o **operador de indireção**, cujo símbolo é *****
- ▶ Se p é um ponteiro, então $*p$ representa o objeto para o qual p aponta no momento

Operadores de endereçamento e de indireção

- ▶ A declaração de uma variável ponteiro reserva um espaço na memória para um ponteiro mas não a faz apontar para um objeto
- ▶ É **crucial** inicializar um ponteiro antes de usá-lo
- ▶ Uma forma de inicializar um ponteiro é atribuir-lhe o endereço de alguma variável usando o operador `&`

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

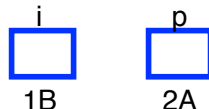
Operadores de endereçamento e de indireção



Operadores de endereçamento e de indireção

- ▶ É possível inicializar uma variável ponteiro no momento de sua declaração:

```
int i, *p = &i;  
i = 2;
```



- ▶ Uma vez que uma variável ponteiro aponta para um objeto, podemos usar o operador de indireção `*` para acessar o valor armazenado no objeto
- ▶ Se `p` aponta para `i`, por exemplo, podemos imprimir o valor de `i` de forma indireta

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

`*p`

- ▶ A função `printf` mostrará o valor de `i` e não o seu endereço

Operadores de endereçamento e de indireção

- ▶ Aplicar o operador `&` a uma variável produz um ponteiro para a variável e aplicar o operador `*` para um ponteiro retoma o valor original da variável

é o mesmo que

```
j = *&i;
```

```
j = i;
```



Operadores de endereçamento e de indireção

- ▶ Enquanto dizemos que p **aponta para** i , dizemos também que $*p$ é um **apelido** para i
- ▶ Não apenas $*p$ tem o mesmo valor que i , mas alterar o valor de $*p$ altera também o valor de i
- ▶ Sempre “traduzir” os operadores unários de endereço $\&$ e de indireção $*$ para *endereço da variável* e *conteúdo da variável apontada por*, respectivamente

Operadores de endereçamento e de indireção

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

```
int main(void)
{
```

```
    char c, *p;
```

```
    p = &c;
```

```
    c = 'a';
```

```
    printf("&c = %p
```

```
    printf("&p = %p
```

```
    c = '/';
```

```
    printf("&c = %p
```

```
    printf("&p = %p
```

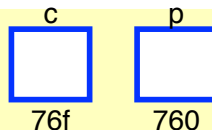
```
    *p = 'Z';
```

```
    printf("&c = %p
```

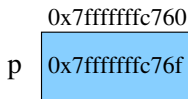
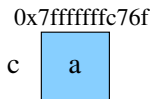
```
    printf("&p = %p
```

```
    return 0;
```

```
}
```



Operadores de endereçamento e de indireção



(a)



(b)

Operadores de endereçamento e de indireção

<code>&c = 0x7fffffffcc76f</code>	<code>c = a</code>	
<code>&p = 0x7fffffffcc760</code>	<code>p = 0x7fffffffcc76f</code>	<code>*p = a</code>
<code>&c = 0x7fffffffcc76f</code>	<code>c = /</code>	
<code>&p = 0x7fffffffcc760</code>	<code>p = 0x7fffffffcc76f</code>	<code>*p = /</code>
<code>&c = 0x7fffffffcc76f</code>	<code>c = Z</code>	
<code>&p = 0x7fffffffcc760</code>	<code>p = 0x7fffffffcc76f</code>	<code>*p = Z</code>

Operadores de endereçamento e de indireção

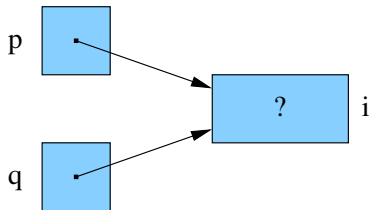
- ▶ A linguagem C permite ainda que o operador de atribuição copie ponteiros, supondo que possuam o mesmo tipo

```
int i, j, *p, *q;
```

```
p = &i;
```

```
q = p;
```

Operadores de endereçamento e de indireção



Ponteiros em expressões

- ▶ Ponteiros podem ser usados em expressões aritméticas de mesmo tipo que seus tipos referenciados
- ▶ Os operadores `&` e `*`, por serem operadores unários, têm precedência sobre os operadores binários das expressões aritméticas em que se envolvem

Ponteiros em expressões

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

```
int main(void)
```

```
{
```

```
    int i, j, *p1, *p2;
```

```
    p1 = &i;
```

```
    i = 5;
```

```
    j = 2 * *p1 + 3;
```

```
    p2 = p1;
```

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

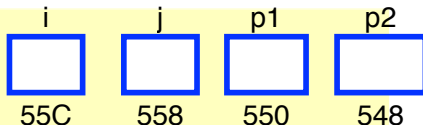
```
    printf("j = %d, &j = %p\n\n", j, &j);
```

```
    printf("&p1 = %p, p1 = %p, *p1 = %d\n", &p1, p1, *p1);
```

```
    printf("&p2 = %p, p2 = %p, *p2 = %d\n\n", &p2, p2, *p2);
```

```
    return 0;
```

```
}
```



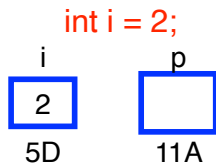
Ponteiros em expressões

```
i = 5, &i = 0x7fffffff55c  
j = 13, &j = 0x7fffffff558
```

```
&p1 = 0x7fffffff550, p1 = 0x7fffffff55c, *p1 = 5  
&p2 = 0x7fffffff548, p2 = 0x7fffffff55c, *p2 = 5
```

9.1 Se i é uma variável e p é uma variável ponteiro que aponta para i , quais das seguintes expressões são apelidos para i ?

- (a) $*p$
- (b) $\&p$
- (c) $*\&p$
- (d) $\&*p$
- (e) $*i$
- (f) $\&i$
- (g) $*\&i$
- (h) $\&*i$

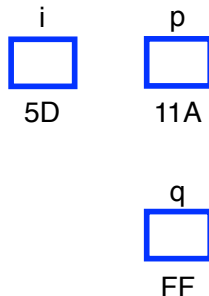


variavel	tipo
i	int
p	int *
*p	int

Exercícios

9.2 Se i é uma variável do tipo `int` e p e q são ponteiros para `int`, quais das seguintes atribuições são corretas?

- (a) `p = i;`
- (b) `*p = &i;`
- (c) `&p = q;`
- (d) `p = &q;`
- (e) `p = *&q;`
- (f) `p = q;`
- (g) `p = *q;`
- (h) `*p = q;`
- (i) `*p = *q;`



`int i = 2;`

`int *p;`

`int *q;`

9.3 Entenda o que o programa abaixo faz, simulando sua execução passo a passo. Depois disso, implemente-o.

```
#include <stdio.h>

int main(void)
{
    int a, b, *pt1, *pt2;

    pt1 = &a;
    pt2 = &b;
    a = 1;
    (*pt1)++;
    b = a + *pt1;
    *pt2 = *pt1 * *pt2;
    printf("a=%d, b=%d, *pt1=%d, *pt2=%d\n", a, b, *pt1, *pt2);

    return 0;
}
```

9.4 Entenda o que o programa abaixo faz, simulando sua execução passo a passo. Depois disso, implemente-o.

```
#include <stdio.h>

int main(void)
{
    int a, b, c, *ptr;

    a = 3;
    b = 7;
    printf("a=%d, b=%d\n", a, b);
    ptr = &a;
    c = *ptr;
    ptr = &b;
    a = *ptr;
    ptr = &c;
    b = *ptr;
    printf("a=%d, b=%d\n", a, b);

    return 0;
}
```

9.5 Entenda o que o programa abaixo faz, simulando sua execução passo a passo. Depois disso, implemente-o.

```
#include <stdio.h>

int main(void)
{
    int i, j, *p, *q;

    p = &i;
    q = p;
    *p = 1;
    printf("i=%d, *p=%d, *q=%d\n", i, *p, *q);
    q = &j;
    i = 6;
    *q = *p;
    printf("i=%d, j=%d, *p=%d, *q=%d\n", i, j, *p, *q);

    return 0;
}
```