Introdução aos ponteiros

Aula 9

Diego Padilha Rubert

Faculdade de Computação
Universidade Federal de Mato Grosso do Sul

Algoritmos e Programação II

Conteúdo da aula

Introdução

- 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.

Introdução

- 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.

Introdução

- 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 sequencialmente a partir de 0 (zero), são chamados de **endereços de memória**

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 sequencialmente a partir de 0 (zero), são chamados de **endereços de memória**

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 sequencialmente a partir de 0 (zero), são chamados de **endereços de memória**

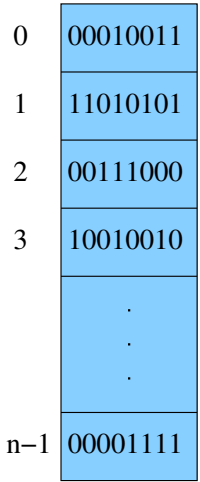
Variáveis ponteiros

- **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 sequencialmente a partir de 0 (zero), são chamados de **endereços de memória**

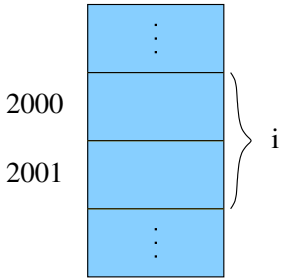
Variáveis ponteiros

- **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 sequencialmente a partir de 0 (zero), são chamados de **endereços de memória**

Variáveis ponteiros



Variáveis ponteiros



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*

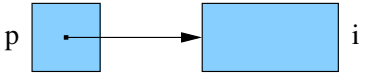
Variáveis ponteiros

- 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

- 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

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 $\star$
- Se p é um ponteiro, então $\star p$ representa o objeto para o qual p aponta no momento

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 $\star$
- Se p é um ponteiro, então $\star p$ representa o objeto para o qual p aponta no momento

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

- 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

- 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;
```

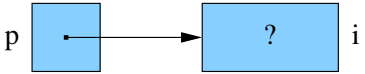
Operadores de endereçamento e de indireção

- 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;
```

- 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);
```

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

Operadores de endereçamento e de indireção

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

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

- 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);
```

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

Operadores de endereçamento e de indireção

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

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

- 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);
```

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

Operadores de endereçamento e de indireção

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

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

- 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);
```

- 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

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

é o mesmo que

```
j = i;
```

Operadores de endereçamento e de indireção

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

Operadores de endereçamento e de indireção

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

Operadores de endereçamento e de indireção

- Enquanto dizemos que *p* **aponta para** *i*, dizemos também que $\star p$ é um **apelido** para *i*
- Não apenas $\star p$ tem o mesmo valor que *i*, mas alterar o valor de $\star p$ 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    c = %c\n", &c, c);
   printf("&p = %p    p = %p    *p = %c\n\n", &p, p, *p);

   c = '/';
   printf("&c = %p    c = %c\n", &c, c);
   printf("&p = %p    p = %p    *p = %c\n\n", &p, p, *p);

   *p = 'Z';
   printf("&c = %p    c = %c\n", &c, c);
   printf("&p = %p    p = %p    *p = %c\n\n", &p, p, *p);

   return 0;
}
```

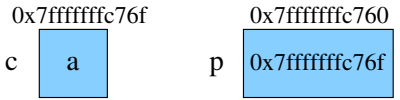
Operadores de endereçamento e de indireção

```
&c = 0x7fffffffc76f    c = a
&p = 0x7fffffffc760    p = 0x7fffffffc76f    *p = a

&c = 0x7fffffffc76f    c = /
&p = 0x7fffffffc760    p = 0x7fffffffc76f    *p = /

&c = 0x7fffffffc76f    c = Z
&p = 0x7fffffffc760    p = 0x7fffffffc76f    *p = Z
```

Operadores de endereçamento e de indireção



(a)

(b)

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

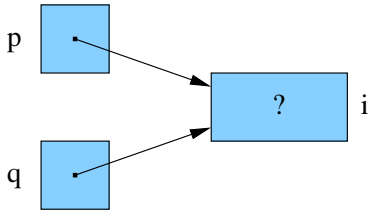
- A linguagem C permite ainda que o operador de atribuição copie ponteiros, supondo que possuam o mesmo tipo

```
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

- 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

```c
#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 = 0x7fffffffc55c
j = 13, &j = 0x7fffffffc558

&p1 = 0x7fffffffc550, p1 = 0x7fffffffc55c, *p1 = 5
&p2 = 0x7fffffffc548, p2 = 0x7fffffffc55c, *p2 = 5
```

Exercícios

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) $\star p$
   (b) $\&p$
   (c) $\star\&p$
   (d) $\&\star p$
   (e) $\star i$
   (f) $\&i$
   (g) $\star\&i$
   (h) $\&\star i$

Exercícios

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;`

Exercícios

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;
}
```

Exercícios

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

```c
#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;
}
```

Exercícios

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;
}
```