

Faculdade de Computação - FACOM

Bacharelado em Sistemas de Informação

FACOM32201 - Algoritmos e Programação II

Prof. Thiago Pirola Ribeiro

Ponteiros

Ponteiros

- Na aula anterior, foram apresentados alguns tipos de dados em C:
 - `int`, `float`, `double`, `char`, `unsigned int`
 - `struct`
- Utilizou-se o operador `&` para trabalhar com endereços de memória, e também variáveis `unsigned int` para guardar esses endereços.
 - Essa não é a forma adequada para trabalhar com endereços.
 - Em C utiliza-se ponteiros!

Ponteiros

- Ponteiro é um tipo de dado que serve para armazenar endereços de memória;

Ponteiros

- Ponteiro é um tipo de dado que serve para armazenar endereços de memória;
- Um ponteiro é uma variável como qualquer outra do programa;

Ponteiros

- Ponteiro é um tipo de dado que serve para armazenar endereços de memória;
- Um ponteiro é uma variável como qualquer outra do programa;
- A diferença é que ela não armazena um valor inteiro, real, caractere ou booleano.

Ponteiros

- Ponteiro é um tipo de dado que serve para armazenar endereços de memória;
- Um ponteiro é uma variável como qualquer outra do programa;
- A diferença é que ela não armazena um valor inteiro, real, caractere ou booleano.
- Serve para armazenar endereços de memória (que, no fundo, são valores inteiros sem sinal, como um `unsigned int` ou `unsigned long long int`).

Ponteiros

- Para declarar uma variável do tipo ponteiro, use a seguinte sintaxe:

```
tipo_de_dado *nome_da_variável
```

- Note que não existe um tipo de dado chamado `pointer`
- O que define o ponteiro é o sinal de `*` juntamente com um outro tipo de dado do programa;
- Esse `tipo_de_dado` deve ser definido, pois o ponteiro armazena um endereço de memória, e precisa especificar qual o tipo de dado que existe naquele endereço que ele armazena.

Ponteiros

- `tipo_de_dado *nome_da_variável`
- Exemplo de Ponteiros que são usados para guardar endereços de variáveis inteiras.

```
int *p;  
int *proximo;  
int *anterior;  
int *abacaxi;
```

- Note que, a única diferença na declaração de uma variável do tipo ponteiro é a adição de um símbolo `*`.

Ponteiros

```
int *p;
```

- O espaço ocupado por um ponteiro em um sistema 32 bits é 4 bytes (o mesmo que um `unsigned int`)
- Sistemas 64 bits os ponteiros ocupam 8 bytes (o mesmo que um `unsigned long long int`)
- Lembre que o ponteiro também é uma variável, e portanto ocupa um espaço de memória;

- Mais exemplos:

```
double *valores;  
double *estoque;
```

```
char *nome;  
char *endereco;
```

```
float *temperatura;
```

Ponteiros – Espaço Ocupado

- Mais exemplos:

```
// armazena endereço de variáveis double
double *valores;
double *estoque;
```

```
// armazena endereço de variáveis char
char *nome;
char *endereco;
```

```
// armazena endereço de variáveis float
float *temperatura;
```

Ponteiros – Espaço Ocupado

- Como o ponteiro armazena um endereço, ele ocupa uma quantidade de memória independente do tipo que ele aponta

```
double *valores;      // 4 bytes  
double *estoque;      // 4 bytes
```

```
char *nome;           // 4 bytes  
char *endereco;       // 4 bytes
```

```
float *temperatura;   // 4 bytes
```

- * Valores para sistemas 32 bits. Sistemas 64 bits os ponteiros ocupam 8 bytes.

Ponteiros

- Como um ponteiro serve para receber um endereço de memória, faz-se, por exemplo, a seguinte operação:

```
int a = 40; // cria uma variável do tipo inteiro, chamada a,  
           //e inicializa com valor 40
```

```
int *p; // cria uma variável do tipo ponteiro para inteiro,  
        //chamada p, e o conteúdo inicial é lixo
```

```
p = &a; // faz p receber o endereço de a.  
        // Dizemos que p aponta para a
```

Mapa de memória

- Sabendo que um ponteiro ocupa 4 bytes (sistema de 32 bits):

```
int a = 40;
```

45			
46			
47		a	int
48	40		
49			
50			
51			
52			
53			
54			
55			
56			

Mapa de memória

- Sabendo que um ponteiro ocupa 4 bytes (sistema de 32 bits):

```
int a = 40;
```

```
int *p;
```

45			
46			
47	40	a	int
48			
49			
50			
51	lx	p	int *
52			
53			
54			
55			
56			

Mapa de memória

- Sabendo que um ponteiro ocupa 4 bytes (sistema de 32 bits):

```
int a = 40;
```

```
int *p;
```

```
p = &a;
```

45			
46			
47	40	a	int
48			
49			
50			
51	47	p	int *
52			
53			
54			
55			
56			

Ponteiros

- Agora que sabemos o que são ponteiros, podemos dizer que o operador `&` retorna um ponteiro para o objeto a qual ele é aplicado

```
p = &a; // é criado um ponteiro para a (&a), que  
       // em seguida é copiado para p
```

```
printf("valor: %d", *(&a)); // mostra o valor de a
```

Mapa de memória

- “p aponta para a”

45			
46			
47	40	a	int
48			
49			
50			
51	47	p	int *
52			
53			
54			
55			
56			

Dereferencing um ponteiro

- Existem operadores específicos em C para trabalhar com ponteiros.
- Um desses operadores é o símbolo *
 - Note que o mesmo símbolo é usado para declarar um ponteiro e também para multiplicação - mas essas operações não são relacionadas.
- O operador * serve para deferenciar (*dereferencing*) um ponteiro - ou seja, ele retorna o conteúdo do endereço de memória que ele referencia/aponta.
- Ao usar o operador *, o tipo retornado será o mesmo tipo apontado pelo ponteiro.

Dereferencing um ponteiro

Exemplo

```
int a = 40; // cria uma variável do tipo inteiro, chamada a,
           //e inicializa com valor 40

int *p; // cria uma variável do tipo ponteiro para inteiro,
        //chamada p, e o conteúdo inicial é lixo

p = &a; // faz p receber o endereço de a.
        // Dizemos que p aponta para a

printf("\n O valor da variavel 'a' eh: %d", *p);
```

Saída:

O valor da variavel 'a' eh: 40

Dereferencing um ponteiro

Exemplo

```
printf("\n O valor da variavel 'a' eh: %d", *p);
```

- O programa vai até o ponteiro **p** e verifica para qual endereço ele aponta.
 - No exemplo, é o endereço 47
- Em seguida, o programa vai até o endereço 47 e busca a informação que está lá.
 - No caso, o valor contido no endereço 47 é o valor inteiro 40, que é mostrado como resposta no printf.

45			
46			
47	40	a	int
48			
49			
50			
51	47	p	int *
52			
53			
54			
55			
56			

Diferentes operadores

Exemplo

```
printf("\n O valor da variavel 'a' eh: %d", *p);  
printf("\n O valor da variavel 'p' eh: %p", p);  
printf("\n O endereço da var. 'p' eh: %p", &p);
```

45			
46			
47	40	a	int
48			
49			
50			
51	47	p	int *
52			
53			
54			
55			
56			

Diferentes operadores

Exemplo

```
printf("\n O valor da variavel 'a' eh: %d", *p);
```

Saida: O valor da variavel 'a' eh: 40

```
printf("\n O valor da variavel 'p' eh: %p", p);
```

```
printf("\n O endereco da var. 'p' eh: %p", &p);
```

45			
46			
47	40	a	int
48			
49			
50			
51	47	p	int *
52			
53			
54			
55			
56			

Diferentes operadores

Exemplo

```
printf("\n O valor da variavel 'a' eh: %d", *p);
```

```
printf("\n O valor da variavel 'p' eh: %p", p);
```

Saida: O valor da variavel 'p' eh: 47

```
printf("\n O endereco da var. 'p' eh: %p", &p);
```

45			
46			
47	40	a	int
48			
49			
50			
51	47	p	int *
52			
53			
54			
55			
56			

Diferentes operadores

Exemplo

```
printf("\n O valor da variavel 'a' eh: %d", *p);  
printf("\n O valor da variavel 'p' eh: %p", p);  
printf("\n O endereco da var. 'p' eh: %p", &p);
```

Saida: O endereco da variavel 'p' eh: 51

45			
46			
47	40	a	int
48			
49			
50			
51	47	p	int *
52			
53			
54			
55			
56			

Dereferencing um ponteiro

- O *Dereferencing* * pode ser usado para atribuição de valores às variáveis apontadas pelos ponteiros.

```
int a = 40;
int *p;
p = &a; // faz p receber o endereço de a.

*p = 59 // altera o conteúdo do endereço apontado por p

printf("\n O valor da variavel 'a' eh: %d", a);
```

Dereferencing um ponteiro

- O *Dereferencing* * pode ser usado para atribuição de valores às variáveis apontadas pelos ponteiros.

```
int a = 40;
int *p;
p = &a; // faz p receber o endereço de a.

*p = 59 // altera o conteúdo do endereço apontado por p

printf("\n O valor da variavel 'a' eh: %d", a);
```

Saída:

O valor da variavel 'a' eh: 59

Dereferencing um ponteiro

```
int a = 40;  
int *p;  
p = &a; // faz p receber o endereço de a.  
*p = 59 // altera o conteúdo do endereço apontado por p
```

45			
46			
47		a	int
48	40		
49			
50			
51		p	int *
52	47		
53			
54			
55			
56			



45			
46			
47		a	int
48	59		
49			
50			
51		p	int *
52	47		
53			
54			
55			
56			

Exemplos

```
// declarando as variáveis
```

```
double val;
```

```
float k;
```

```
// declarando os ponteiros
```

```
double *pval;
```

```
float *pk;
```

```
// atribuindo os valores das variáveis aos ponteiros
```

```
pval = &val;
```

```
pk = &k;
```

```
// alterando o conteúdo das variáveis via ponteiros
```

```
*pval = 33.45;
```

```
*pk = 2.4;
```

Exemplos

```
// declarando as variáveis
```

```
double val;
```

```
float k;
```

```
// declarando os ponteiros
```

```
double *pval;
```

```
float *pk;
```

Parou Aqui !!!

```
// atribuindo os valores das variáveis aos ponteiros
```

```
pval = &val;
```

```
pk = &k;
```

```
// alterando o conteúdo das variáveis via ponteiros
```

```
*pval = 33.45;
```

```
*pk = 2.4;
```

4			
5	lx	val	double
6			
7			
8			
9			
10			
11			
12			
13	lx	k	float
14			
15			
16			
17	lx	pval	*double
18			
19			
20			
21	lx	pk	*float
22			
23			
24			
25			

Exemplos

```
// declarando as variáveis
```

```
double val;
```

```
float k;
```

```
// declarando os ponteiros
```

```
double *pval;
```

```
float *pk;
```

```
// atribuindo os valores das variáveis aos ponteiros
```

```
pval = &val;
```

```
pk = &k;
```

Parou Aqui !!!

```
// alterando o conteúdo das variáveis via ponteiros
```

```
*pval = 33.45;
```

```
*pk = 2.4;
```

4			
5	lx	val	double
6			
7			
8			
9			
10			
11			
12			
13	lx	k	float
14			
15			
16	5	pval	*double
17			
18			
19			
20	13	pk	*float
21			
22			
23			
24			
25			

Exemplos

```
// declarando as variáveis
double val;
float k;

// declarando os ponteiros
double *pval;
float *pk;

// atribuindo os valores das variáveis aos ponteiros
pval = &val;
pk = &k;

// alterando o conteúdo das variáveis via ponteiros
*pval = 33.45;
*pk = 2.4;
```

Parou Aqui !!!

4			
5	33.45	val	double
6			
7			
8			
9			
10			
11			
12			
13	2.4	k	float
14			
15			
16			
17	5	pval	*double
18			
19			
20			
21	13	pk	*float
22			
23			
24			
25			

Está correto?

```
1 double *preco;  
2  
3 *preco = 50.0;
```

Está correto?

```
1 double *preco;  
2  
3 *preco = 50.0;
```

- Não houve alocação para guardar um número `double`
- Houve somente alocação para guardar um ponteiro para `double`

Está correto?

```
1 double *preco;  
2 Parou Aqui !!!  
3  
4 *preco = 50.0;
```

68			
69	lx	preco	*double
70			
71			
72			
73			
74			

Está correto?

```
1 double *preco;
```

```
2 Parou Aqui !!!
```

```
3
```

```
4 *preco = 50.0;
```

- Ao declarar uma variável, o conteúdo dela é lixo.

68			
69	lx	preco	*double
70			
71			
72			
73			
74			

Está correto?

```
1 double *preco;
```

```
2 Parou Aqui !!!
```

```
4 *preco = 50.0;
```

- O programa tentará alterar o que está no endereço 13, e poderá travar.
- Pode alterar outras variáveis.

67			
68			
69		preco	*double
70	13		
71			
72			
73			
74			

-			
9			
10			
11			
12			
13		k	float
14	2.4		
15			
16			
17		pval	*double
18	5		
19			
20			
21		pk	*float
22	13		
23			
24			
25			

Está correto?

```
1 double *preco;
```

```
2 Parou Aqui !!!
```

```
3  
4 *preco = 50.0;
```

- O programa tentará alterar o que está no endereço 13, e poderá travar.
- Pode alterar outras variáveis.

Problema!!!

Memória invadida (8 bytes pois preço é `*double`)

67			
68			
69		preco	*double
70	13		
71			
72			
73			
74			

-			
9			
10			
11			
12			
13		k	float
14	2.4		
15			
16			
17		pval	*double
18	5		
19			
20			
21		pk	*float
22	13		
23			
24			
25			

Inicialização

- Um ponteiro pode ter o valor especial NULL que é o endereço de “nenhum lugar”.

Ex: `int *p = NULL;`

- Pode-se usar o valor 0 (zero) ao invés de NULL

Endereço	Blocos (1 byte)	Nome variável	Tipo
0 / NULL	indefinido	----	----
1			
2			

28			
29			
30	NULL	p	*int
31			
32			
33			
34			

Mudar para Ponteiro

```
1  int k;
2  unsigned int endereco_de_k;
3
4  // inicializando k
5  k = 10;
6  printf("\n Valor da variavel 'k': %d \n",k);
7
8  // obtendo o endereço da variável 'k' // usando o operador &
9  endereco_de_k = &k;
10
11 printf("\n Endereco da variavel 'k': %u \n",endereco_de_k);
12 printf("\n Endereco da variavel 'k': %u \n",&k);
13 printf("\n Endereco da variavel 'k': %p (em hexadecimal)\n",&k);
14
15 // sabemos que o scanf pede um endereço de memória
16 // o que acontece se passarmos o endereço da
17 // variável k?
18 printf("\n Digite o valor novo valor, a ser armazenado no endereço da variavel k: ");
19 // OBSERVE que não estamos usando & no scanf! Isso porque já temos o endereço
20 scanf("%d",endereco_de_k);
21
22 // mostrando o novo valor de 'k'
23 printf("\n\n Valor da variavel k, apos scanf de 'endereco_de_k': %d \n",k);
24
25 // mostrando o endereço de 'k' que deve permanecer o mesmo de antes
26 printf("\n Endereco da variavel 'k': %u \n",endereco_de_k);
27 printf("\n Endereco da variavel 'k': %u \n",&k);
28 printf("\n Endereco da variavel 'k': %p (em hexadecimal)\n",&k);
```

Ponteiro

```
1  int k;
2  int *endereco_de_k;
3
4  // inicializando k
5  k = 10;
6  printf("\n Valor da variavel 'k': %d \n",k);
7
8  // obtendo o endereço da variável 'k' // usando o operador &
9  endereco_de_k = &k;
10
11 printf("\n Endereco da variavel 'k': %u \n", endereco_de_k);
12 printf("\n Endereco da variavel 'k': %u \n",&k);
13 printf("\n Endereco da variavel 'k': %p (em hexadecimal)\n",&k);
14
15 // sabemos que o scanf pede um endereço de memória
16 // o que acontece se passarmos o endereço da
17 // variável k?
18 printf("\n Digite o valor novo valor, a ser armazenado no endereço da variavel k: ");
19 // OBSERVE que não estamos usando & no scanf! Isso porque já temos o endereço
20 scanf("%d", endereco_de_k);
21
22 // mostrando o novo valor de 'k'
23 printf("\n\n Valor da variavel k, apos scanf de 'endereco_de_k': %d \n",k);
24
25 // mostrando o endereço de 'k' que deve permanecer o mesmo de antes
26 printf("\n Endereco da variavel 'k': %u \n", endereco_de_k);
27 printf("\n Endereco da variavel 'k': %u \n",&k);
28 printf("\n Endereco da variavel 'k': %p (em hexadecimal)\n",&k);
```

Exercícios

- Escreva um programa que contenha duas variáveis inteiras. Leia essas variáveis do teclado. Em seguida, exiba o conteúdo alocado no maior endereço.
- Escreva um programa que declare um inteiro, um real e um char, e ponteiros para inteiro, real, e char. Associe as variáveis aos ponteiros (use &). Modifique os valores de cada variável usando os ponteiros. Mostre na tela os valores das variáveis antes e após a modificação.

Faculdade de Computação - FACOM

Bacharelado em Sistemas de Informação

Prof. Thiago Pirola Ribeiro