Ponteiros

Prof. Jefferson T. Oliva

Algoritmos e Estrutura de Dados I (AE22CP)
Engenharia de Computação
Departamento Acadêmico de Informática (Dainf)
Universidade Tecnológica Federal do Paraná (UTFPR)
Campus Pato Branco

DAINF
DEPARTAMENTO ACADÊMICO DE INFORMÁTICA
CÂMPUS PATO BRANCO

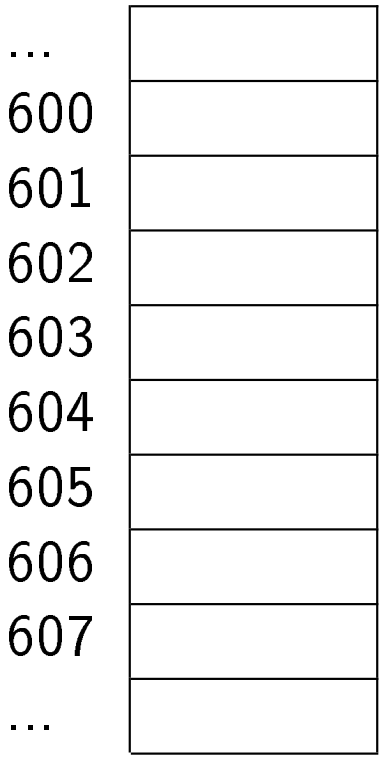
UTFPR
UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

Sumário

Sumário

Variáveis

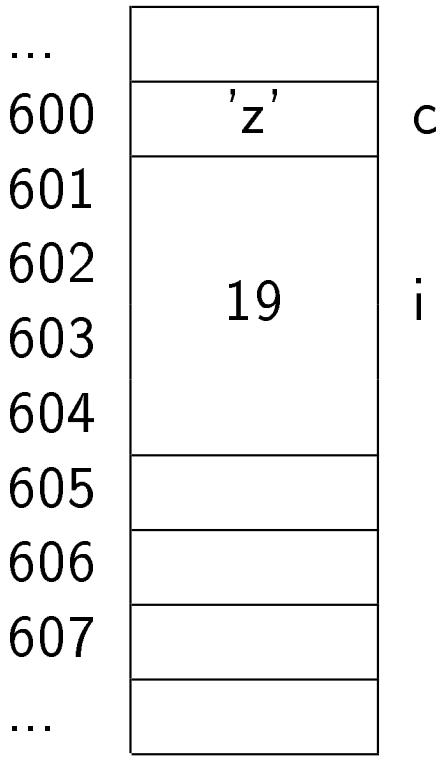
- Variáveis armazenam dados na memória
  - int, float, double, char, struct



Variáveis

- Variáveis armazenam dados na memória

```
char c = 'z'; // 1 byte
int i = 19; // 4 bytes
```



- Nesse exemplo, podemos dizer que o endereço de *c* é 600 e o de *i* é 601

Variáveis

- Obter o tamanho de tipos de dados
  - Função *sizeof*

```
typedef struct retangulo{
   int x, y;
}Retangulo;

int main(void){
   printf("%d\n", sizeof(double));
   printf("%d\n", sizeof(int));
   printf("%d\n", sizeof(char));
   printf("%d\n", sizeof(Retangulo));

   return 0;
}
```

Variáveis

- Obter o endereço de variáveis
  - Operador &

```
int main(void){
   int i = 19;
   char c = 'z';
   printf("Valor de i:  %d\n", i);
   printf("Endereco de i:  %i\n", &i);
   printf("Endereco de i (em hexadecimal):  %x\n", &i);
   printf("Valor de c:  %c\n", c);
   printf("Endereco de c:  %i\n", &c);
   printf("End.  de c em hexadecimal:  %x\n", &c);

   return 0;
}
```

  - No *scanf*, os argumentos básicos são o tipo e o endereço da variável

Sumário

Ponteiros

- Ponteiro é uma variável que contém um endereço de memória
- Esse endereço pode ser a posição de uma outra variável na memória
- Cada ponteiro pode armazenar o endereço do seu respectivo tipo de dado
- Se um ponteiro contém o endereço de uma variável, então é dito que esse ponteiro aponta para a variável

Ponteiros

- Os ponteiros são similares aos tipos de dados
- O ponteiro possui uma declaração especial
  - O nome da variável deve ser precedido pelo operador *
    - tipo *nome;
      ou
    - tipo* nome;
      ou
    - tipo * nome;
  - Exemplos:

```
int *pi;

char *pc;
```

Ponteiros

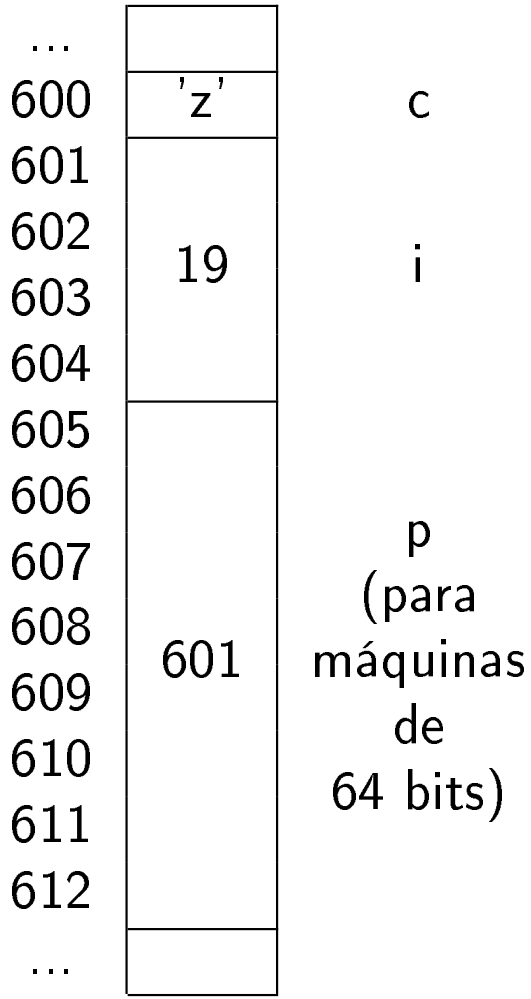
Operadores

- & ("o endereço de")
  - Retorna o endereço de memória do operando
  - Exemplo: $a = \&b;$
- * ("valor de")
  - Declarar variável do tipo ponteiro
  - Acessa o valor armazenado no endereço apontado
  - exemplo: $q = *p;$
    - q recebe o valor que está no endereço p

Ponteiros

Operadores

```
char c = 'z';
int i = 19;
int *p = &i;
```



Ponteiros

Operadores

```
int main(void){
   int i;
   int *p_i;

   i = 30;
   p_i = &i;
   *p_i = 15;
   printf("%d\n", i);
   printf("%d\n", *p_i);
   printf("%d\n", &i);
   printf("%d\n", p_i);

   return 0;
}
```

```
int main(void){
   int *p_i;

   *p_i = 30; // Não pode.  Por quê?

   return 0;
}
```

Ponteiros

Passagem de parâmetros

```
void troca(int a, int b){
    int aux;

    aux = a;
    a = b;
    b = aux;
}

int main(void){
    int x, y;

    x = 5;
    y = 10;

    troca(x, y);

    printf("%d\n", x);
    printf("%d\n", y);

    return 0;
}
```

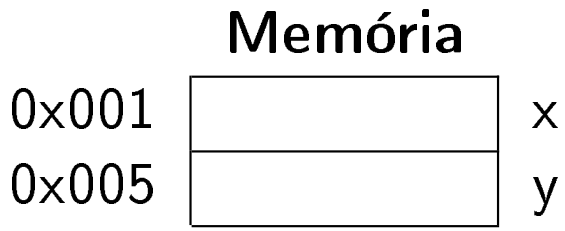
- O código irá compilar?
- É de fato realizada uma troca de valores entre x e y?

Ponteiros

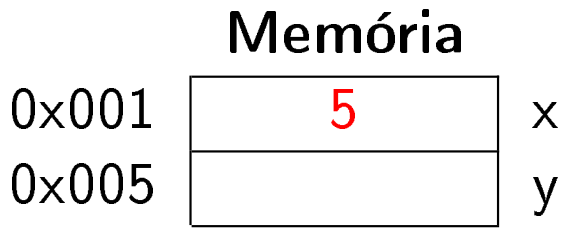
Passagem de parâmetros

```
void troca(int *a, int *b){
   int aux;

   aux = *a;
   *a = *b;
   *b = aux;
}

int main(void){
   int x, y;

   x = 5;
   y = 10;

   troca(&x, &y);

   printf("%d\n", x);
   printf("%d\n", y);

   return 0;
}
```

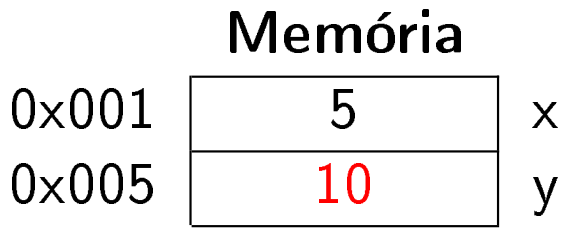
```
int main(void){
   int x, y;
   x = 5;
   y = 10;
   troca(&x, &y);
   printf("%d\n", x);
   printf("%d\n", y);

   return 0;
}
```



```
int main(void){
   int x, y;
   x = 5;
   y = 10;
   troca(&x, &y);
   printf("%d\n", x);
   printf("%d\n", y);

   return 0;
}
```



```
int main(void){
   int x, y;
   x = 5;
   y = 10;
   troca(&x, &y);
   printf("%d\n", x);
   printf("%d\n", y);

   return 0;
}
```



```
int main(void){
   int x, y;
   x = 5;
   y = 10;
   troca(&x, &y);
   printf("%d\n", x);
   printf("%d\n", y);

   return 0;
}
```

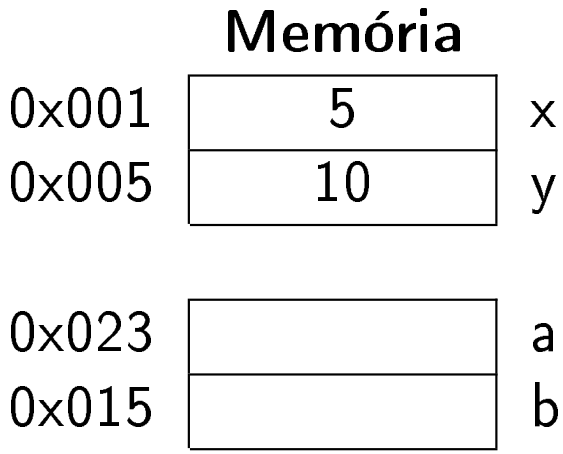
Memória

0x001	5	x
0x005	10	y

Em *troca(&x, &y), a função chamadora entra em modo de espera*

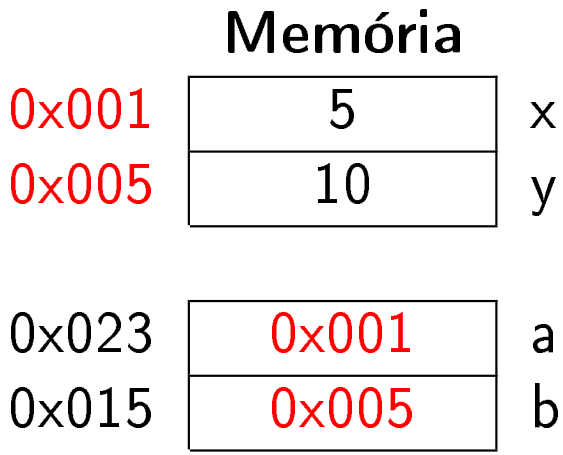
A função *troca* é iniciada declarando os parâmetros *a* e *b*

```
void troca(int *a, int *b){
   int aux;

   aux = *a;
   *a = *b;
   *b = aux;
}
```

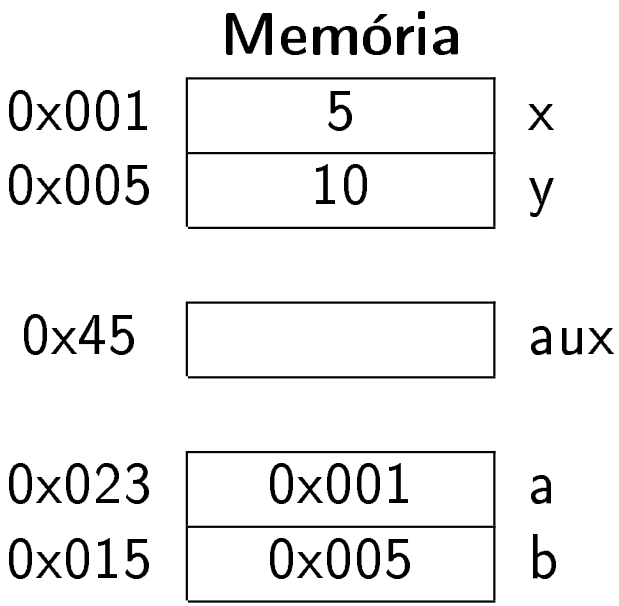


- A função *troca* é inicializada com os valores recebidos

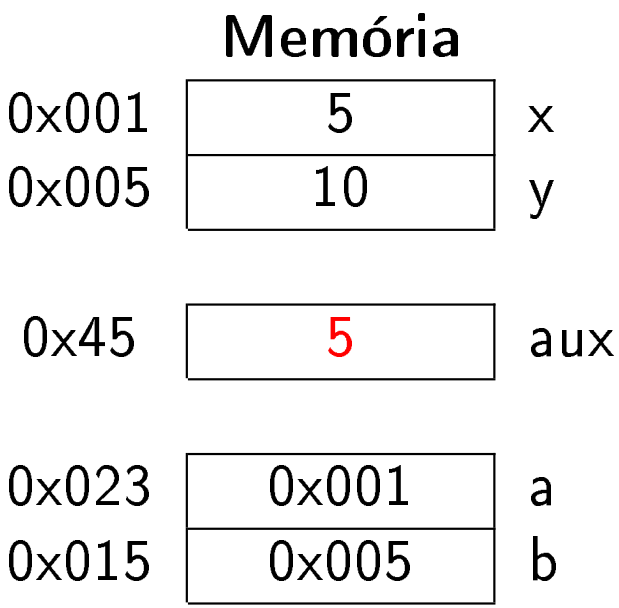
```
void troca(int *a, int *b){
   int aux;

   aux = *a;
   *a = *b;
   *b = aux;
}
```



```
void troca(int *a, int *b){
   int aux;

   aux = *a;
   *a = *b;
   *b = aux;
}
```

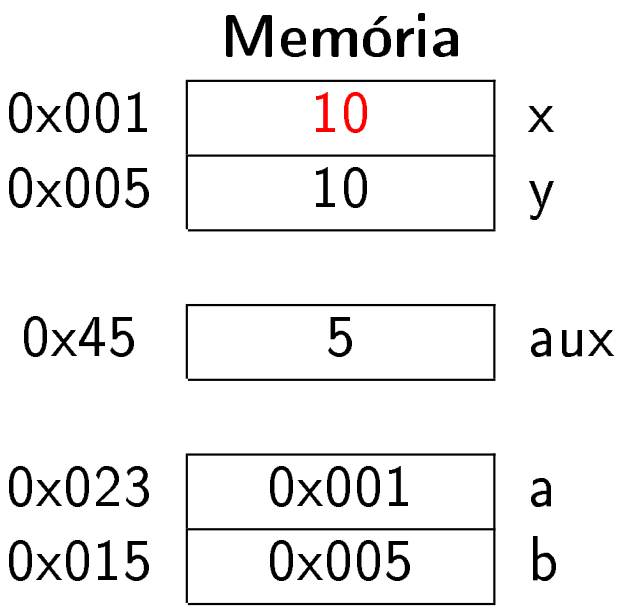


```
void troca(int *a, int *b){
   int aux;

   aux = *a;
   *a = *b;
   *b = aux;
}
```



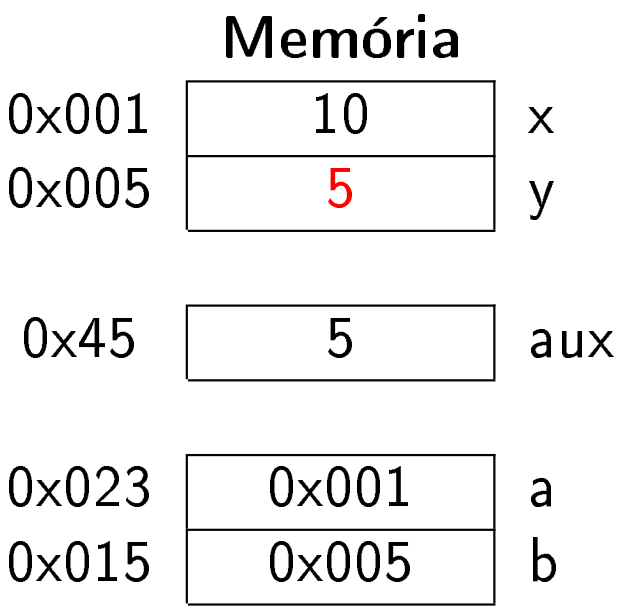
O valor dentro do endereço que *a* armazena é acessado

```
void troca(int *a, int *b){
   int aux;

   aux = *a;
   *a = *b;
   *b = aux;
}
```

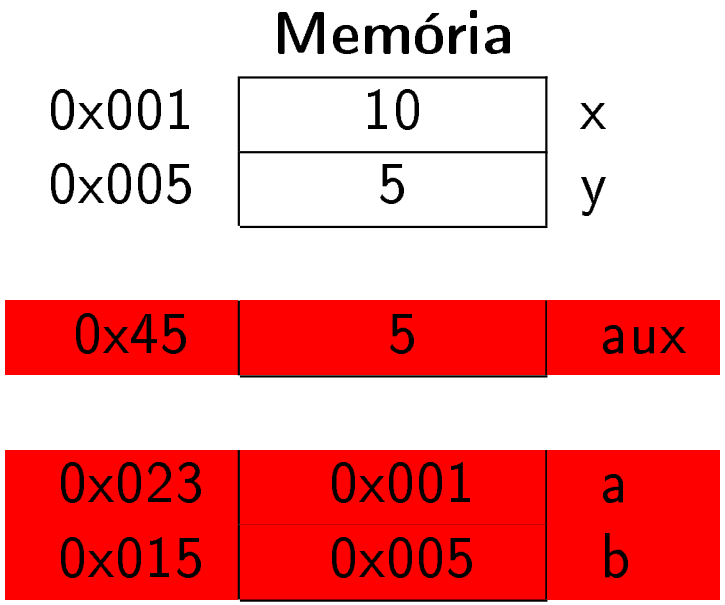


Os valores dentro dos endereços que *a* e *b* armazenam são acessados

```
void troca(int *a, int *b){
   int aux;

   aux = *a;
   *a = *b;
   *b = aux;
}
```



```
void troca(int *a, int *b){
   int aux;

   aux = *a;
   *a = *b;
   *b = aux;
}
```



A execução da função é finalizada e as variáveis locais são descartadas

```
int main(void){
   int x, y;
   x = 5;
   y = 10;
   troca(&x, &y);
   printf("%d\n", x);
   printf("%d\n", y);

   return 0;
}
```

Memória

0x001	10	x
0x005	5	y

Função chamadora volta à execução

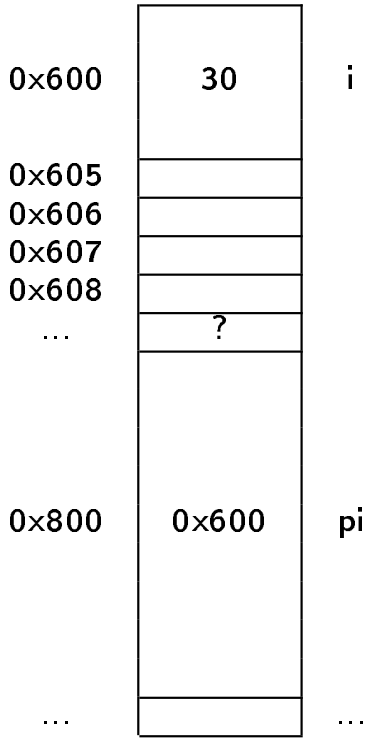
Após, o conteúdo atualizado de x e de y é impresso

- A adição e a subtração em ponteiros devem ser feitos cuidadosamente: pode acarretar em acesso indevido aos espaços de memória

```
char c = '0';
int i = 0;

char *pc;
int *pi;

pc = &c;
pi = &i;

pc++; // desloca 1 byte
pi++; // desloca 4 bytes
```

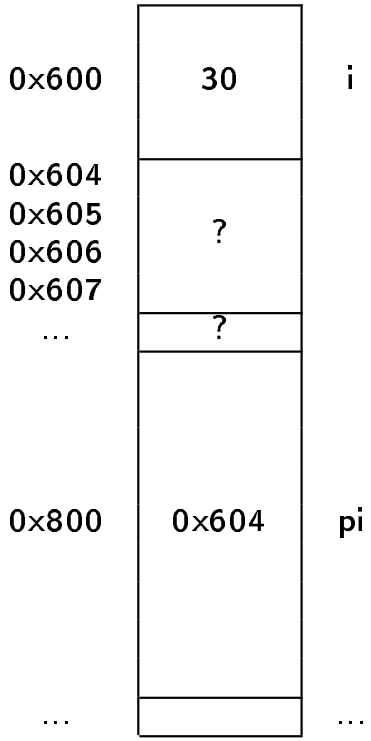
- A adição e a subtração em ponteiros devem ser feitos cuidadosamente: pode acarretar em acesso indevido aos espaços de memória
  - Exemplo com int e ponteiro de int

```
int i = 30;
int *pi;
pi = &i;
pi++; // desloca 4 bytes
```



- A adição e a subtração em ponteiros devem ser feitos cuidadosamente: pode acarretar em acesso indevido aos espaços de memória
  - Exemplo com int e ponteiro de int

```
int i = 30;
int *pi;
pi = &i;
pi++; // desloca 4 bytes
```



- A comparação entre ponteiros também é válida, desde que os mesmos princípios dos dados numéricos sejam seguidos

```
int main(void){
   char a, b;
   char *pa, *pb;

   pa = &a;
   pb = &b;

   if (pa == pb)
     printf("Enderecos iguais.\n");
   else
     printf("Enderecos diferentes.\n");

   return 0;
}
```

- Vetores
  - Vetores são referenciados por um nome e pelos índices
  - O nome de um vetor é o ponteiro para o primeiro índice
  - Os índices informam a quantidade (*e.g.* i * tamanho do tipo da variável) de deslocamentos a partir da posição zero

- Vetores

```
int main(void){
   int i;
   int v[5];
   *v = 0;
   *(v + 1) = 10;
   *(v + 2) = 20;
   *(v + 3) = 30;
   *(v + 4) = 40;

   for (i = 0; i < 5; i++)
     printf("v[%d] = %d\n", i, *(v + i));

   return 0;
}
```

- Vetores

```
int main(void){
   int i;
   int v[5];
   v[0] = 0;
   v[1] = 10;
   v[2] = 20;
   v[3] = 30;
   v[4] = 40;

   for (i = 0; i < 5; i++)
     printf("v[%d] = %d\n", i, v[i]);

   return 0;
}
```

  - Operações aritméticas do ponteiro respeitam o tamanho do tipo de dado

Ponteiros Genéricos

Ponteiros Genéricos

- Pode apontar para todos os tipos de dados existentes
- Ponteiro do tipo *void**

```
int main(void){
   char c = 'Q';
   char *pc;
   void *p;

   pc = &c;
   p = &c;
   printf("Char:  %c\n", *pc);
   printf("ponteiro:  %p\n", p);

   p = &pc; // endereço do ponteiro
   printf("Char:  %c\n", *pc);
   printf("ponteiro:  %p\n", p);

   return 0;
}
```

Ponteiros Genéricos

- Para acessar ao valor do endereço genérico, deve ser utilizado um modificador de tipo (*cast*)

```
int main(void){
   char c = 'Q';
   void *p;

   p = &c;
   printf("Char:  %c\n", c);
   printf("ponteiro:  %c\n", *(char*) p);

   return 0;
}
```

Ponteiros Genéricos

- Para acessar ao valor do endereço genérico, deve ser utilizado um modificador de tipo (*cast*) [2]

```
int main(void){
   char c = 'Q';
   char *pc;
   void *p;

   pc = (char*) p;
   printf("Char:  %c\n", c);
   printf("ponteiro:  %c\n", *pc);

   return 0;
}
```

Ponteiros Genéricos

- A definição do tipo do ponteiro genérico necessita de cuidados:
  - Para acessar valores sempre deve se converter para o tipo de ponteiro utilizado (*cast*)
  - As operações aritméticas sempre utilizam 1 byte

Ponteiros

- Vantagens
  - Maior liberdade na manipulação de variáveis e no uso de funções
  - Uso de alocação dinâmica de memória
- Principal desvantagem
  - Possibilidade de acessar posições não alocadas ou indevidas

Exercício

- No código a seguir, o que será impresso em tela?

```
int main(void){
   int x[4];
   int *a, *b;
   *x = 118;
   *(x + 2) = 4;
   *(x + x[2] - 1) = 51;
   x[1] = 25;
   a = &x;
   b = &x[2];
   printf(" %d\n", x[0]);
   printf(" %d\n", x[1]);
   printf(" %d\n", x[2]);
   printf(" %d\n", x[3]);
   printf(" %d\n", *a);
   printf(" %d\n", *b);
   return 0;
}
```

Exercício

- No código a seguir, o que será impresso em tela?
  - Resposta: 118, 25, 4, 51, 118, 4

Erros Comuns em Ponteiros

```
int main(){
   int *p;
   *p = 10;
    return 0;
}
```

- Atribuição a um endereço relacionado a um ponteiro nulo
- Solução 1: alocação dinâmica (assunto em uma das próximas aulas)
- Solução 2: ponteiro apontar para o endereço de uma variável

```
int main(){
   int *p;
   int b;
   p = &b;
   *p = 10;
    return 0;
}
```

- Solução 3: remover os asteriscos (não usar ponteiro)

Erros Comuns em Ponteiros

```
int main(){
   int *p;
   int b;
   p = b;
    return 0;
}
```

- O ponteiro está recebendo uma variável em vez de endereço
- Solução 1: *int p = &b;*
- Solução 2: usar o operador * ("valor de"), mas o endereço não pode ser nulo

```
int main(){
   int *p;
   int b;
   int c;
   p = &b;
   *p = c;
    return 0;
}
```

Erros Comuns em Ponteiros

```
int main(){
   int *p;
   int b = 10, c;
   p = &b;
   c = p;
    return 0;
}
```

- A variável está recebendo valor do endereço, em vez do valor que está nessa localização (não consta erro de compilação, mas alerta (*warning*))

- Solução: usar o operador *

```
int main(){
   int *p;
   int b = 10, c;
   p = &b;
   c = *p;
    return 0;
}
```

Erros Comuns em Ponteiros

```
int main(){
   int *p;
   float b;
   p = &b;
    return 0;
}
```

- O ponteiro do tipo int está apontando para um endereço do tipo float, por mais que não ocorra erro de compilação, ainda haverá mensagem de alerta

- Solução: ponteiros devem apontar para endereços do mesmo tipo

```
int main(){
   int *p;
   int b;
   p = &b;
    return 0;
}
```

Erros Comuns em Ponteiros

```
void troca(int *a, int *b){
   int aux;
   aux = *a;
   *a = *b;
   *b = aux;
}
int main(void){
   int x = 5, y = 10;
   x = 5;
   y = 10;
   troca(x, y);
   return 0;
}
```

- A função *troca* deveria receber ponteiro, mas recebeu variáveis
- Solução 1: passar o endereço das variáveis à função
- Solução 2: passar ponteiros à função

Erros Comuns em Ponteiros

- Solução 1:

```
void troca(int *a, int *b){
   int aux;
   aux = *a;
   *a = *b;
   *b = aux;
}
int main(void){
   int x = 5, y = 10;
   troca(&x, &y);
   return 0;
}
```

Erros Comuns em Ponteiros

- Solução 2:

```
void troca(int *a, int *b){
   int aux;
   aux = *a;
   *a = *b;
   *b = aux;
}
int main(void){
   int x = 5, y = 10;
   int *px = &x; // ou int *px; px = &x;
   int *py = &y;
   troca(px, py);
   return 0;
}
```

Erros Comuns em Ponteiros

- Solução 3 (bônus):

```
void troca(int *a, int *b){
   int aux;
   aux = *a;
   *a = *b;
   *b = aux;
}
int main(void){
   int x = 5, y = 10;
   int *py = &y;
   troca(&x, py);
   return 0;
}
```

Referências I


Arakaki, R.; Arakaki, J.; Angerami, P. M.; Aoki, O. L.; Salles, D. S.
Fundamentos de programação C: técnicas e aplicações.
LTC, 1990.


Deitel, H. M.; Deitel, P. J.
Como programar em C.
LTC, 1999.


Pereira, S. L.
Estrutura de Dados e em C: uma abordagem didática.
Saraiva, 2016.


Tenenbaum, A.; Langsam, Y.
Estruturas de Dados usando C.
Pearson, 1995.