

Ponteiros

- É uma variável que ao invés de guardar valores, guarda posições de memória (guarda endereços)

Declaração:

* = Referência

- Variável comum: int a;
- Ponteiro: int *b // int é o tipo da variável que vai apontar
- & = Aponta para o endereço
- a = 2;
- b = &a ↳ armazena o endereço de a
- Notação hexadecimal (o endereço)
- Quando imprime b = endereço, quando imprime *b = o que tem dentro do endereço (conteúdo)
- *b = 3 no № endereço de b o valor será 3, ou seja, o valor de a vai mudar para 3 (pq b aponta para a)
- Pode apontar pra um lugar e modificá-lo
- b = &c Importante
↳ referência (endereço)
- *b = 5 Importante
↳ de-referência (permite modificar a variável do endereço)
↳ Contendo

Sobre ponteiros:

- 1) Diga a seguinte sequência de instruções em um programa C:

int *pti; int i=10; pti=&i;

Qual alternativa é falsa?

- a) pti armazena o endereço de i
 - b) *pti é igual a 10
 - c) se executar *pti=20; i passará a ter o valor 20
 - d) se alterar o valor de i, *pti será modificado
- (1) pti é igual a 10

Comentários sobre a questão:

- a) f) & indica endereço

- a) Sim, quando se tem * há relação com o conteúdo que está no endereço
- b) Sim, mesma coisa da letra b, * = conteúdo que está no endereço
- c) Sim, o valor de *pti está relacionado com o valor de i
- d) Não, o valor de pti é o endereço de i (seja constante se fosse o valor de *pti)

2) Quais os valores de x, y e p ao final do trecho de código abaixo?

```

int x, y; // declaração
int *p;
y = 0; // y = 0
p = &y; // p = Endereço de y (hexadecimal)
x = *p; // x = 0 (conteúdo de y (pq p faz referência a y))
x = y; // x = 4
(*p)++; // y = 1 (0+1) e *p = 1
--x; // x = 3
(*p) += x; *p = 1 + 3 = 4 (y = 4 e *p = 4)
Resposta = x = 3, y = 4 e p = &y (endereço de y)
    
```

3) Responda o trecho de programa abaixo:

```

int i = 99; // i = 99
int j;
int *p;
p = &i // p é o endereço de i
j = *p + 100; // 99 + 100 = 199 (i + 100)
    
```

(l) valor de j ao final deste é:

j = 199 ~~x~~

4) Diga o que é feito pelo programa abaixo:

```
int a = 5; // a = 5  
int b = 12; // b = 12  
int c;  
int *p;  
int *q;  
p = &a; // p = &a  
q = &b; // q = &b  
c = *p + *q; // *p = 5 *q = 12 c = 5 + 12 = 17
```

5) Diga o que é feito pelo código:

```
int a = 25; // a = 25  
int *pa = &a // pa = endereço de a  
int b = *pa + a // 25 + 25 = 50
```

O valor de b ao final deste é:

Esse $*pa$ ter um endereço atribuído me deixou confusa
(deu na mesma $*pa$ mesmo valendo um endereço, vale o seu conteúdo)

Vetor e ponteiros (aritmética de vetores)

- int vetor [3]
 - ↳ ponteiro para o inicio do vetor
- Se der cout na variável vetor aparece o endereço
- Vetor é um ponteiro (aponta para a primeira posição (vetor[0]))
- $*vetor$ = Mostra o conteúdo da posição 0
- Como fazer apontar para a próxima posição? $p++$
- O código vetor - ponteiros explica isso
- Se fizer $p = p + 2$ pula 2 posições (não precisa se limitar a $p++$), pode usar $p--$ também (esse $p++$ tem haver com deslocamento)

Marcos - 10

Modelo ponteiro

1) Diga o seguinte trecho de programa:

```
float x[3];
```

```
float y;
```

```
float *p;
```

$y = 4; \quad // y = 4$

$p = \&y; \quad // p = endereço de y$

```
for (int i = 0; i < 3; i++) { \quad // vai passar 3x
    x[i] = *p + i; \quad // 1º = 4.0 = 4    2º 4.1 = 5    3º 4.2 = 6
```

Qual conteúdo de x após a execução deste programa será?

\$ 0,4,84 ✓

2) Diga o trecho de código abaixo:

```
int *p;
```

```
int x[10];
```

$p = x; \quad // p vale x, x é um vetor de 10 posições$

```
cout << (int)*p; \quad //
```

```
p = &x[2];
```

```
cout << (int)*p;
```

Se o primeiro cout foi 10, qual será o segundo?

Explicação do professor:

O que será exibido pelo cout são endereços de memória, o ponteiro p armazena endereços de memória, o primeiro cout está exibindo o endereço de memória 10 que no caso, é o endereço da primeira posição do vetor x ($p = x$), no segundo cout, seria exibido o endereço de memória da segunda posição do vetor x, mas $p = \&x[2]$. Como a primeira posição está no endereço 10, a segunda estará duas posições depois, uma variável intira ocupa normalmente 4 bytes então o endereço da segunda posição seria a posição 18 ($10 + 2 * 4$) (10. Valores da posição zero, 2 = Quantos "casas" tem, 4 = bytes (próximo)

Alocação dinâmica de memória

- Criar vetor do tamanho que quiser
- *vetor = NULL (não aponta pra lugar nenhum)

Etapas:

- 1) Set o tamanho desejado para o vetor
 - 2) Bloco memória para o vetor // operador new
 - 3) Preencher o vetor
 - 4) Mostrar o vetor
- Código alocações - memória explica isso
 - vetor = new int [tam] Aloca memória vetor armazena tam inteiros

- O operador new que faz a mágica
- Precisa colocar o comando delete (é pra liberar memória RAM) // libera memória, só usa delete quando usa new

Alocação dinâmica de memória com struct

- isso vai ser usado no lugar de
- Tem que ter com a struct ter virado ponteiro
- Exemplo:** produto → preco (agora a struct é um ponteiro)
- No código struct - alocação - memória só é com um produto e isso será usado em uma matéria futuramente que vai juntar tudo
 - Stack = Memória usada pelo programa
 - Heap = Quando aloca (por isso usar remove)

Alocar memória para um vetor com struct

- Precisa saber tamanho para o vetor
- Deve voltar a usar

produto [i] → código X
produto [i].codigo ✓

- dem contado ()
- dado produto [tam] \times (tamanho só é declarado em produto = meu dado [tam'])

Passagem de parâmetros para funções

→ Por valor:

// retorna $x + y$

```
void soma ( int x, int y, int z) { // não retorna nada
    z = x + y;
}
```

↳ Void pq o resultado vai estar no z

Deu erro por causa do c
(meio que some a função z)

int main () {
 int a,b; // elementos a serem somados
 int z; // a + b
 cin >> a >> b; // ler a e b
 soma (a,b,c); // calcular a + b a=x b=y
 cout << "soma = " << z; // mostrar o resultado

→ Por referência:

```
void soma ( int x, int y, int *z) {
    *z = x + y; // z sendo um ponteiro
```

int main () {
 int a = 2, b = 3, c = 0; // usa ponteiro
 soma (a,b,&c); // endereço do c = *z
 cout << a << b << c << endl;
 return 0;

→ Mudando passar a ponteiro → alteração da fun

função para main

→ feito main simples
void soma (int x, int y, int & z)
// únicas alterações

