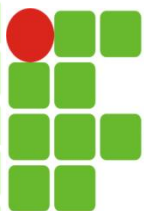




INSTITUTO FEDERAL DE
EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
SUL-RIO-GRANDENSE
Campus Passo Fundo

ALGORITMOS II

Prof. Adilso Nunes de Souza



ROTEIRO

- **Aritmética de ponteiro**
- **Ponteiro para ponteiro**



ARITMÉTICA DE PONTEIRO

- Ao pensar em aritmética de ponteiros é necessário conhecer o tamanho em bytes de cada tipo de dado:
 - Char: 1 byte
 - Int: 4 bytes
 - Float: 4 bytes
 - Double: 8 bytes
- O operador `sizeof()` pode ser utilizado para identificar o tamanho em bytes que uma variável ocupa ou um tipo de dado.



ARITMÉTICA DE PONTEIRO

```
int x = 0;  
cout << sizeof(char) << endl;  
cout << sizeof(x) << endl;  
cout << sizeof(float) << endl;  
cout << sizeof(double) << endl;
```

Saída:

1
4
4
8



ARITMÉTICA DE PONTEIRO

- Outra observação importante na manipulação de ponteiros é a ordem de precedência dos operadores.

Ex.

```
int x = 234;  
int *px;  
px = &x;  
(*px)++;  
cout << *px << endl;
```

Saída

235



ARITMÉTICA DE PONTEIRO

- Só é permitido a utilização de quatro operadores aritméticos com ponteiros:
 - Incremento (++)
 - Decremento (--)
 - Adição (+)
 - Subtração (-)



ARITMÉTICA DE PONTEIRO

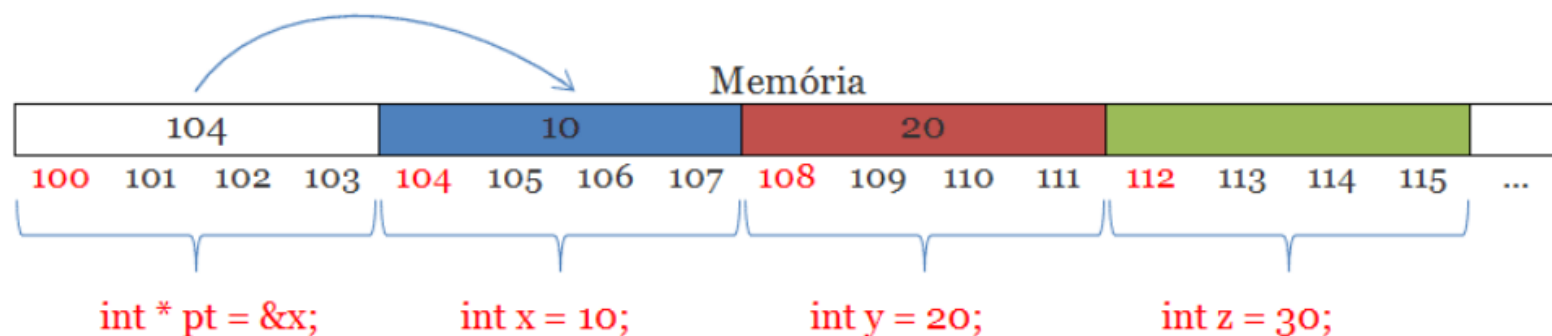
```
int x;  
int *px;  
px = &x;  
px++;
```

- Supondo que o endereço de memória de X é 100, ao incrementar o ponteiro px ele passa a referenciar o endereço de memória 104, ou seja o próximo endereço inteiro. O mesmo vale para o decremento.



ARITMÉTICA DE PONTEIRO

Conhecendo as operações básicas de incremento e decremento de ponteiros, precisamos entender o que ocorre com o endereço armazenado no ponteiro quando uma das operações aritméticas é utilizada.

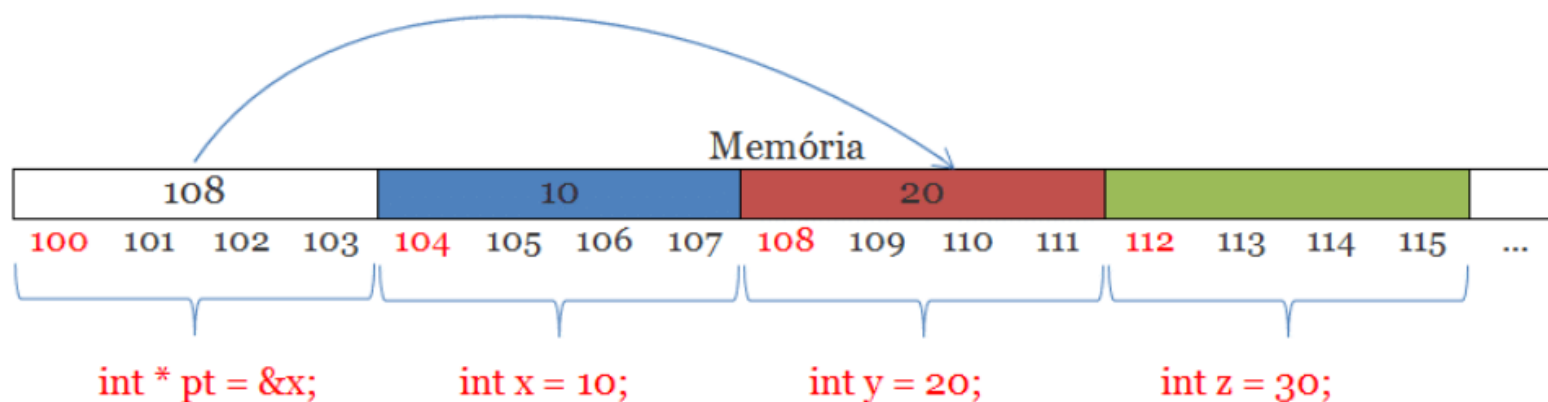




ARITMÉTICA DE PONTEIRO

Executando o incremento do ponteiro `pt`:
`pt++;`

O ponteiro passa a apontar para a localização da memória do próximo elemento do seu tipo base, como pode ser observado:





ARITMÉTICA DE PONTEIRO

Também não se limita apenas a operações de incremento e decremento, sendo possível adicionar ou subtrair números inteiros, exemplo:

$$pt = pt + 4;$$

$4 \times \text{sizeof}(\text{int}) = 16 \text{ bytes}$

neste caso o ponteiro desloca-se 16 bytes na memória.



ARITMÉTICA DE PONTEIRO

Nestes exemplos consideramos os endereços de memória especificados, mas na prática os endereços só são consecutivos tratando-se de vetores.

```
int x[5] = {5,6,7,8,9};
int *px, i = 0;
px = x;
while(i < 5)
{
    cout << px << endl; //mostra o endereço atual do px
    cout << *px << endl;
    px++;
    i++;
}
```



ARITMÉTICA DE PONTEIRO

Após incrementar um ponteiro de um array deve-se observar que, ao final do percurso do array, o ponteiro não está mais no endereço onde o vetor inicia.

```
int x[5] = {5,6,7,8,9};
int *px, i = 0;
px = x;
cout << "Endereco inicial do vetor: " << px << endl;
while(i < 5)
{
    cout << px << endl;
    cout << *px << endl;
    px++;
    i++;
}
cout << "Endereco final do vetor: " << px << endl;
```



ARITMÉTICA DE PONTEIRO

Para retornar ao endereço inicial do array basta decrementar, a mesma quantidade de bytes que foram incrementadas.

```
int x[5] = {5,6,7,8,9};
int *px, i = 0;
px = x;
cout << "Endereco inicial do vetor: " << px << endl;
while(i < 5)
{
    cout << px << endl;
    cout << *px << endl;
    px++;
    i++;
}
cout << "Endereco final do vetor: " << px << endl;
px -= i;
cout << "Volta ao endereco inicial: " << px << endl;
```

```
Endereco inicial do vetor: 0x61fe00
0x61fe00
5
0x61fe04
6
0x61fe08
7
0x61fe0c
8
0x61fe10
9
Endereco final do vetor: 0x61fe14
Volta ao endereco inicial: 0x61fe00
```



ARITMÉTICA DE PONTEIRO

- Portanto, sempre que um ponteiro for incrementado, ele apontará para a localização da memória do próximo elemento de seu tipo base.
- Sempre que ele for decrementado, apontará para a localização do elemento anterior de seu tipo base.
- Também é possível adicionar ou subtrair números inteiros a um ponteiros.

$px = px + 5;$

px passa a apontar para o quinto elemento do tipo base além daquele para o qual ele está apontando atualmente.



ARITMÉTICA DE PONTEIRO

- Aritmética de ponteiros pode ser utilizada para acessar elementos de um array, sendo na maioria das vezes mais rápido do que o acesso indexado.
- Atenção especial ao trabalhar com array do tipo char, pois quando o objeto cout recebe como argumento um ponteiro char, imprime o texto armazenado com base neste endereço.



ARITMÉTICA DE PONTEIRO

■ Exemplo:

```
char texto[100];  
char *ptexto;  
ptexto = texto;  
cout << "Digite um texto qualquer: ";  
gets(texto);  
while(*ptexto)  
{  
    cout << ptexto << endl;  
    cout << *ptexto << ", ";  
    ptexto++;  
}
```

```
Digite um texto qualquer: Adilso  
Adilso  
A, dilso  
d, ilso  
i, lso  
l, so  
s, o  
o,
```




ARITMÉTICA DE PONTEIRO

- Também é possível acessar qualquer posição dentro do array, basta para isso indicar a posição a partir do endereço atual.
`cout << *(ptexto + 3);`
- Este exemplo mostra a quarta posição do vetor pois o mesmo inicia em zero.
- Atenção especial no uso dos parênteses, isso se deve pois o operador `*` tem maior prioridade que o de adição, sem os parênteses ele acabaria somando 3 ao elemento atual apontado.



ARITMÉTICA DE PONTEIRO

- A manipulação do tipo string também é possível com o uso de aritmética de ponteiros, para isso basta observar a forma de endereçar o ponteiro.

string texto;

char *ptexto;

ptexto = &texto[0];



PONTEIRO PARA PONTEIRO

- Também conhecido como múltipla indireção, é a capacidade de criarmos ponteiros que apontam para outros ponteiros, podendo atingir o nível que desejar.
- Quando um valor é indiretamente apontado por um ponteiro para um ponteiro, o acesso àquele valor exige que o operador asterisco seja aplicado duas vezes.



PONTEIRO PARA PONTEIRO

```
int x = 6, *px, **px2, ***px3;  
px = &x;  
px2 = &px;  
px3 = &px2;
```

```
cout << *px << endl;  
cout << **px2 << endl;  
cout << ***px3 << endl;
```



PONTEIRO PARA PONTEIRO

■ Exemplo de utilização:

```
void leitura(int *pv);  
void mostra(int **p);
```

```
main()  
{  
    int valor;  
    leitura(&valor);  
    cout << "\nValor no main: " <<  
    valor;  
}
```

```
void leitura(int *pv)  
{  
    cout << "Informe um valor: ";  
    cin >> *pv;  
    fflush(stdin);  
    mostra(&pv);  
}  
  
void mostra(int **p)  
{  
    cout << "\nValor digitado: " <<  
    **p;  
    **p += 3;  
    cout << "\nValor atualizado: "  
    << **p;  
}
```



PONTEIRO PARA PONTEIRO

■ Exemplo de utilização com array:

```
main()
{
    int vet[5], *pvet;
    pvet = vet;
    srand(time(NULL));
    for(int i = 0; i < 5; i++)
    {
        *(pvet + i) = rand() % 10;
    }
    cout << pvet << endl;
    mostra(&pvet);
}
```

```
void mostra(int **p)
{
    //cout << p << endl;
    for(int i = 0; i < 5; i++)
        //cout << (*p)[i] << ", ";
        cout << ((*p)+i) << ", ";
    calcula(&p);
}

void calcula(int ***p2)
{
    //cout << "\n" << p2 << endl;
    //cout << "\n" << *p2 << endl;
    //cout << "\n" << (*(p2)) << endl;
    int soma = 0;
    for(int i = 0; i < 5; i++)
        soma += (*(p2)+ i);
        //soma += (*(p2))[i];
    cout << "\n\nSoma: " << soma;
}
```



REFERÊNCIAS

- SCHILDT, Herbert. C completo e total. Rio de Janeiro: Editora Ciência Moderna LTDA 3ª edição. 2000.
- PEREIRA, Silvio do Lago. Estrutura de Dados Fundamentais: Conceitos e Aplicações, 12. Ed. São Paulo, Érica, 2008.
- LORENZI, Fabiana. MATTOS, Patrícia Noll de. CARVALHO, Tanisi Pereira de. Estrutura de Dados. São Paulo: Ed. Thomson Learning, 2007.
- VELOSO, Paulo. SANTOS, Celso dos. AZEVEDO, Paulo. FURTADO, Antonio. Estrutura de dados. Rio de Janeiro: Ed. Elsevier, 1983 27ª reimpressão.