PONTEIROS

Estrutura de Dados

O que é?

- Ponteiros são variáveis que armazenam o endereço de memória de outras variáveis.
- Dizemos que um ponteiro “aponta” para uma variável quando contém o endereço na memoria RAM da mesma.
- Os ponteiros podem apontar para qualquer tipo de variável: int, float, double, etc.

O que é?

- Cada variável de um programa ocupa um certo número de bytes consecutivos na memória do computador.
- Uma variável do tipo char ocupa 1 byte. Uma variável do tipo int ocupa 4 bytes e um double ocupa 8 bytes em muitos computadores.
- O número exato de bytes de uma variável é dado pelo operador sizeof.

O que é?

- Cada variável (em particular, cada registro e cada vetor) na memória tem um endereço. Na maioria dos computadores, o endereço de uma variável é o endereço do seu primeiro byte. Por exemplo, depois das declarações

Por que usar ponteiros?

- Ponteiros são muito úteis quando uma variável tem que ser acessada em diferentes partes de um programa.
- Caso este dado seja alterado, não há problema algum, pois todas as partes do programa tem um ponteiro que aponta para o endereço onde reside o dado atualizado.

Por que usar ponteiros?

- Existem várias situações onde ponteiros são úteis, por exemplo:
  - Alocação dinâmica de memória
  - Manipulação de vetores e matrizes.
  - Para retornar mais de um valor em uma função.
  - Referência para listas, pilhas, árvores e grafos.

Declaração de ponteiro

- Para declarar uma variável como um ponteiro deve-se utilizar o símbolo ‘*’ entre o tipo e o nome da variável. A forma geral da declaração é:

- **tipo * nome_Ponteiro;**

- No exemplo acima temos o tipo que é o tipo de dado da variável que vamos apontar, podendo ser int, float ou até mesmo uma struct.

Declaração de ponteiro

- Depois temos o * (asterisco) que nesse caso determina que a variável é um ponteiro. E por fim temos “Nome_Ponteiro” que, como o próprio nome diz, é o nome do ponteiro.

```
int * ptr;
int valor = 10;
ptr = &valor;
```

Operadores

- Ao trabalhar com ponteiros fazemos o uso de dois operadores, são eles:
  - o operador de **indireção '*';**
  - o operador **unário '&'** para obter o endereço de uma variável (também chamado de ponteiro constante, pois representa um endereço de memória fixo).

Impressão de Ponteiros

```
int * ptr;
int valor = 10;
ptr = &valor;
printf("Endereço = %x", &valor);
printf("Endereço = %x", ptr);
printf("Valor = %d", *ptr);
```

- `printf("Valor = %d", valor);`

Impressão de Ponteiros

- Resultado:
  - **22ff44 //endereço de &valor**
  - **22ff44 //conteúdo do ponteiro**
  - **10 //valor da variável valor usando o ponteiro**
  - **10 //valor acessadi diretamente**

- Foi utilizado **%x** para exibir o endereço e o conteúdo do ponteiro ptr, pois trata-se de um valor hexadecimal por ser endereço de memória.

Impressão de Ponteiros

- Podemos usar o **%p** para ponteiros também;
  - *ptr – A variável ptr tem o endereço da variável valor, para encontrar esse valor usamos o operador * antes do nome do ponteiro.
  - valor – Estamos explicitamente imprimindo o conteúdo dessa variável valor do tipo inteiro.

Ponteiro para ponteiros

- Desse modo, um ponteiro pode armazenar o endereço de outro ponteiro, ocasionando uma indireção múltipla. Para declarar um ponteiro de indireção múltipla utiliza-se N vezes o operador *, sendo N o nível de indireção. No exemplo abaixo é declarado um ponteiro para ponteiro.

Ponteiro para ponteiros

int ** ptr2; //**ponteiro para ponteiro** do tipo

int * ptr1; //**ponteiro** para o tipo inteiro

```
int var = 10;
ptr2 = &ptr1;
ptr1 = &var;
*ptr1 = 30;
**ptr2 = 50;
printf("Valor ptr1 = %d", *ptr1);
printf("Valor ptr2 = %d", **pptr2);
```

Ponteiro para ponteiros

- Resultado
  - 50
  - 50

- *ptr1: Conteúdo do endereço apontado, ou seja, o conteúdo de ‘var’.
- **ptr2: Acessa o conteúdo do endereço armazenado no ponteiro que é referenciado por ‘ptr2’, ou seja, acessa ‘var’ indiretamente.

Operações com Ponteiros

- Atribuição
  - Se temos dois ponteiros, p1 e p2, e quisermos que p1 aponte para o mesmo lugar que p2, basta fazermos
    - p1=p2.
  - É interessante observar que se o objetivo for que a variável apontada por p1 tenha o mesmo conteúdo da variável apontada por p2 deve-se fazer
    - *p1=*p2.

Operações com Ponteiros

- Incremento e Decremento
  - Quando incrementamos um ponteiro ele passa a apontar para o próximo valor do mesmo tipo para o qual o ponteiro aponta. Esta é uma razão pela qual o compilador pr
  - O decremento funciona de forma semelhante. Supondo que p é um ponteiro, as operações são escritas como: ecisa saber o tipo de um ponteiro.
    - p++;
    - p--;

Operações com Ponteiros

- Estamos falando de operações com ponteiros e não de operações com o conteúdo das variáveis para as quais eles apontam. Por exemplo, para incrementar o conteúdo da variável apontada pelo ponteiro p, faz-se:
  - **(*p)++;**

Operações com Ponteiros

- Soma e Subtração de Inteiros com Ponteiros
  - Vamos supor que você queira incrementar um ponteiro de 15 unidades. Basta fazer:
    - **p=p+15; ou p+=15;**
  - E se você quiser usar o conteúdo da memória apontada 15 posições adiante:
    - ***(p+15);**
- A subtração funciona de forma similar.

Operações com Ponteiros

- Comparação entre dois Ponteiros
  - Podemos saber se dois ponteiros são iguais ou diferentes (== e !=).
  - No caso de operações do tipo >, <, >= e <= estamos comparando qual ponteiro aponta para uma posição mais alta na memória.
  - A comparação entre dois ponteiros se escreve como a comparação entre outras duas variáveis quaisquer

Operações com Ponteiros

- Há entretanto operações que não podemos efetuar sobre um ponteiro.
- Não se pode dividir ou multiplicar ponteiros, adicionar dois ponteiros, adicionar ou subtrair floats ou doubles a ponteiros

Funções - Chamada por Referência

- Quando fazemos a chamada de uma função podemos passar argumentos, caso existam, num processo conhecido como chamada por valor.
- Esse processo é caracterizado por copiar os valores dos argumentos para os parâmetros da função.
- Outro método de passar um valor para função é denominado chamada por referência.
- Nesse procedimento o endereço do argumento é passado para função, logo temos que utilizar ponteiros para manipular os endereços!

Funções - Chamada por Referência

```
void soma10(int x){
  x = x + 10;
  printf("Valor de x apos a soma = %d \n",x);
  return;
}
void soma10p(int *x){
  *x = *x + 10;
  printf("Valor de x apos a soma = %d \n",*x);
  return;
}
```

Funções - Chamada por Referência

```
int main(void)
{
    int numero;
    printf("Digite um numero: ");
    scanf("%d", &numero);
    printf("O numero digitado foi: %d \n",numero);
    soma10(numero); //chamada da função
    printf("Agora o numero vale: %d \n",numero);
    soma10p(&numero); //chamada da função com ponteiro como
    parâmetro
    printf("Agora o numero vale: %d \n",numero);
    return 0;
}
```

Funções - Chamada por Referência

```
void troca_valores(int *ptrx, int *ptry);
int main(void)
{
  int a, b;
  printf("Digite o primeiro valor: ");
  scanf("%d", &a);
  printf("Digite o segundo valor: ");
  scanf("%d", &b);
  printf("Voce digitou os valores na seguinte ordem: %d e %d\n", a, b);
  troca_valores(&a, &b);
 printf("Os valores trocados sao: %d e %d\n", a, b);
  getch();
  return 0;
}
```

Funções - Chamada por Referência

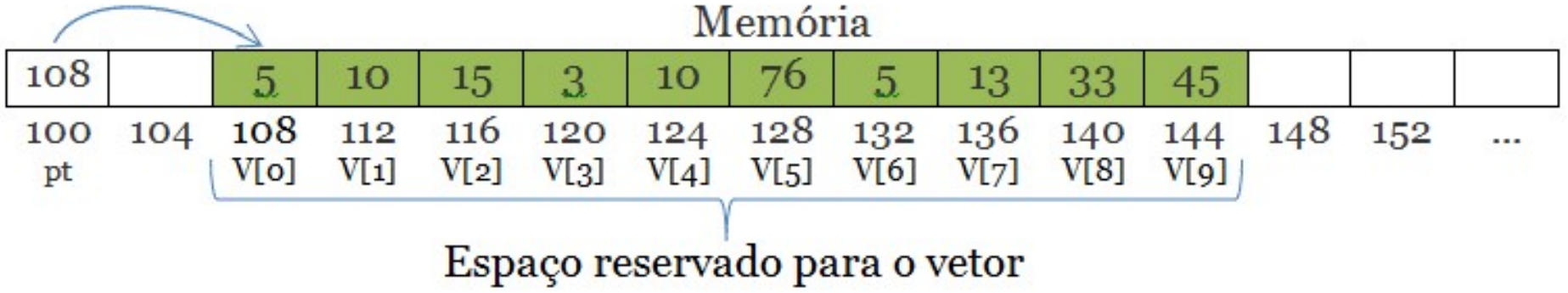
```
void troca_valores(int *ptrx, int *ptry){
        int auxiliar;
        //auxiliar recebe o conteúdo apontado por ptrx
        auxiliar = *ptrx;
  //coloca o valor que está no local apontado por ptry em ptrx
        *ptrx = *ptry;
  //finalmente, ptry recebe o valor armazenado em auxiliar
         *ptry = auxiliar;
         return;
}
```

Ponteiros e Vetores

- Arrays unidimensionais, ou vetores, são um conjunto de dados de mesmo tipo e que são armazenados em posições contíguas da memória [1]. Considere, por exemplo, um vetor de inteiros com 10 elementos armazenados a partir do endereço 108.
  - `int v[] = {5, 10, 15, 3, 10, 76, 5, 13, 33, 45};`
  - `int * pt;`
  - `pt = v; //atribui o endereço do vetor`

Ponteiros e Vetores

- Um ponteiro é criado e passa a apontar para o primeiro elemento do vetor. Para atribuir o endereço de um vetor para um ponteiro basta utilizar o próprio nome do vetor, isto é, o nome representa o endereço do primeiro elemento.

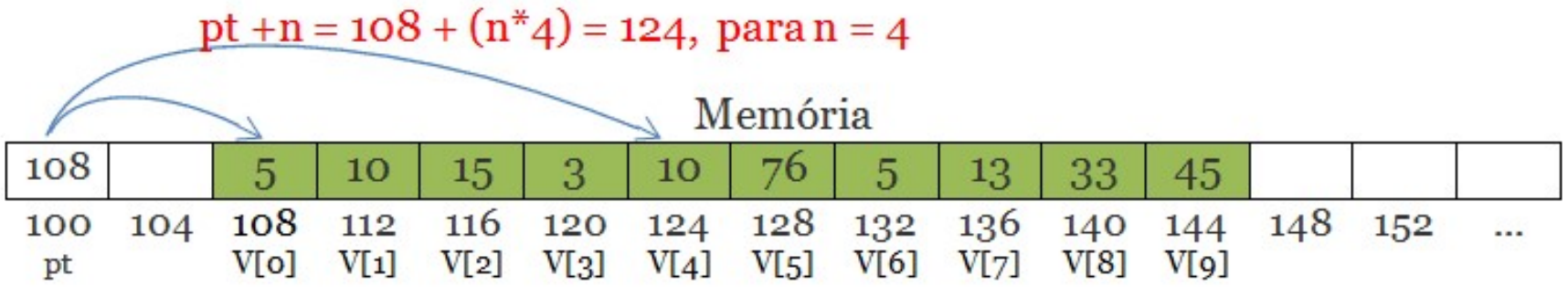


Ponteiros e Vetores

- Para obter o endereço de outro índice é necessário utilizar o operador '&'. Portanto, as duas atribuições mostradas abaixo são equivalentes.
  - pt = v;
  - pt = &v[0];
- Logo, o endereço do quinto elemento pode ser obtido da seguinte forma:
  - pt = &v[4];

Ponteiros e Vetores

- Diante disso, verifica-se que o elemento de um vetor é armazenado numa posição de memória na qual o seu endereço é equivalente à soma do endereço base com o total de bytes dos elementos até a posição desejada. Dito de outra maneira, se pt aponta para o endereço base do vetor então 'V[n]' é equivalente '*(pt + n)'. Na figura 3 é ilustrado essa alteração do endereço apontado.



Ponteiros e Vetores

```
int * pt;
int i;
 pt = v;
 for(i = 0; i < 10; i++)
{
   printf("V[%i] = %i\r\n", i, *(pt + i));
}
```

Ponteiros e Vetores

```
int matrx [10] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };
int *p;
p=matrx;
printf ("O terceiro elemento do vetor e: %d",p[2]);
```

OBS.: Podemos ver que p[2] equivale a *(p+2).

Ponteiros e Vetores

```
int v[] = {5, 10, 15, 3, 10, 76, 5, 13,
33, 45};
int * pt;
int i;
pt = v;
for(i = 0; i < 10; i++)
{
   printf("V[%i] = %i\r\n", i, *pt++);
}
```

Ponteiros e Vetores

- De fato, as duas formas de indexar os elementos do vetor apresentam o mesmo resultado, contudo a aritmética de ponteiros pode ser mais rápida do que a indexação direta, pois na indexação direta o endereço do elemento que será acessado é sempre calculado (somar o endereço base com a posição desejada)

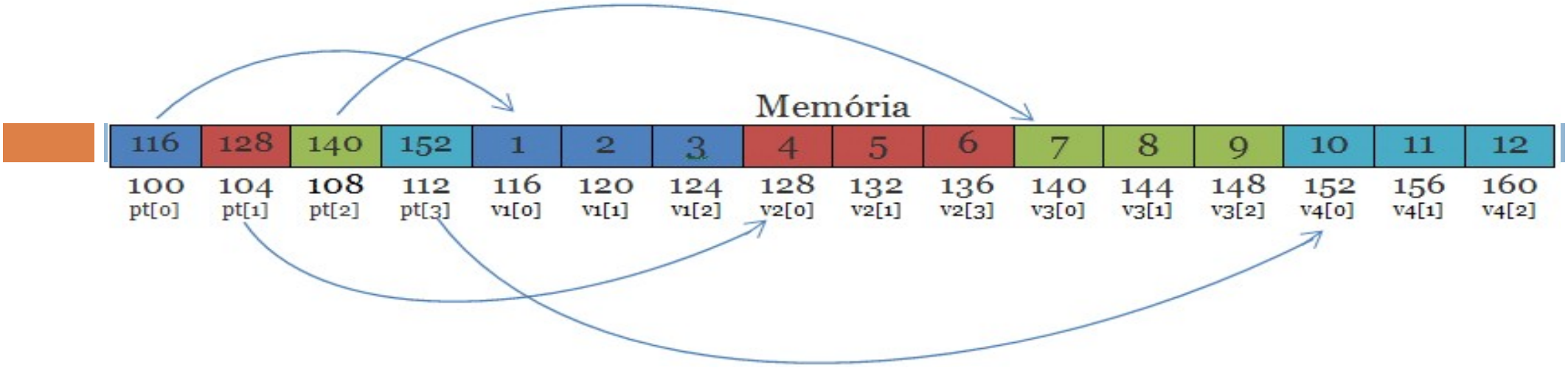
Vetores de Ponteiros

- Os ponteiros também podem ser declarados na forma de vetores. Considere o exemplo abaixo que define um vetor de ponteiros com 4 elementos e mais quatros vetores de 3 elementos.

Vetores de Ponteiros

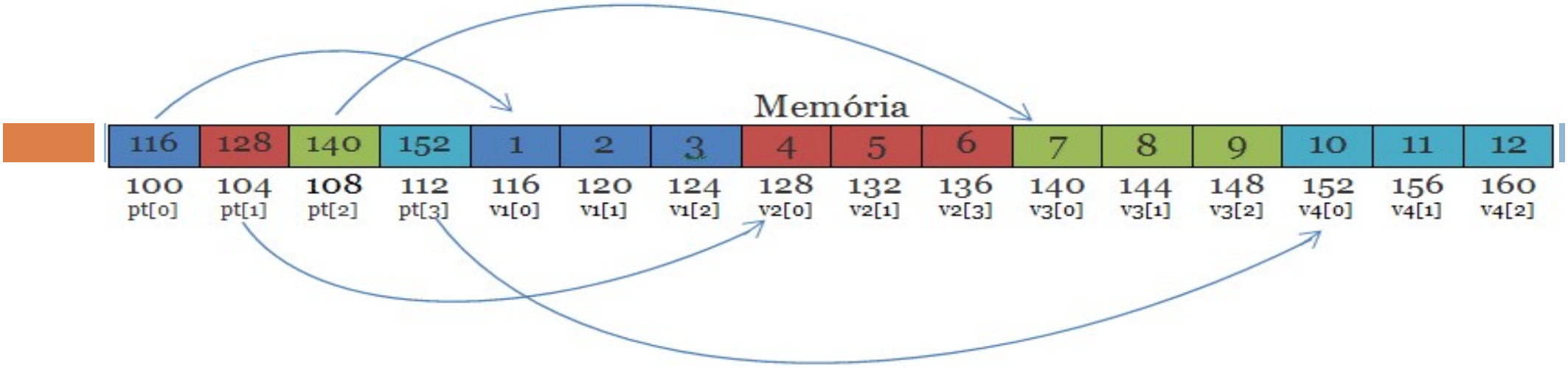
```
int * pt [4]; //vetor de ponteiros do tipo inteiro
int v1[3] = {1, 2, 3}; //vetor 1 com três elementos
int v2[3] = {4, 5, 6}; //vetor 2 com três elementos
int v3[3] = {7, 8, 9}; //vetor 3 com três elementos
int v4[3] = {10, 11, 12}; //vetor 4 com três elementos

pt[0] = v1; //atribui o endereço do vetor1 para o ponteiro pt[0]
pt[1] = v2; //atribui o endereço do vetor2 para o ponteiro pt[1]
pt[2] = v3; //atribui o endereço do vetor3 para o ponteiro pt[2]
pt[3] = v4; //atribui o endereço do vetor4 para o ponteiro pt[3]
```



Agora, é necessário lembrar que ao acessar os elementos pt[0], pt[1], pt[2] e pt[3], estaremos manipulando ponteiros. Para acessar os elementos de cada vetor a partir dos ponteiros basta utilizar o operador '*' e indicar o índice desejado. Considere os casos abaixo:

- *pt[0] é o valor 1, pois estamos acessando o conteúdo do endereço 116, ou seja, v1[0];
- *pt[1] é o valor 4, pois estamos acessando o conteúdo do endereço 128, ou seja, v2[0];
- *pt[2] é o valor 7, pois estamos acessando o conteúdo do endereço 140, ou seja, v3[0];
- *pt[3] é o valor 10, pois estamos acessando o conteúdo do endereço 152, ou seja, v4[0].



- *(*(pt+0)) é o valor 1;
- *(*(pt+1)) é o valor 4;
- *(*(pt+2)) é o valor 7;
- *(*(pt+3)) é o valor 10.