ALGORITMOS E ESTRUTURAS DE DADOS I

Prof. Caio César de Freitas Dantas

Ponteiros e Vetores

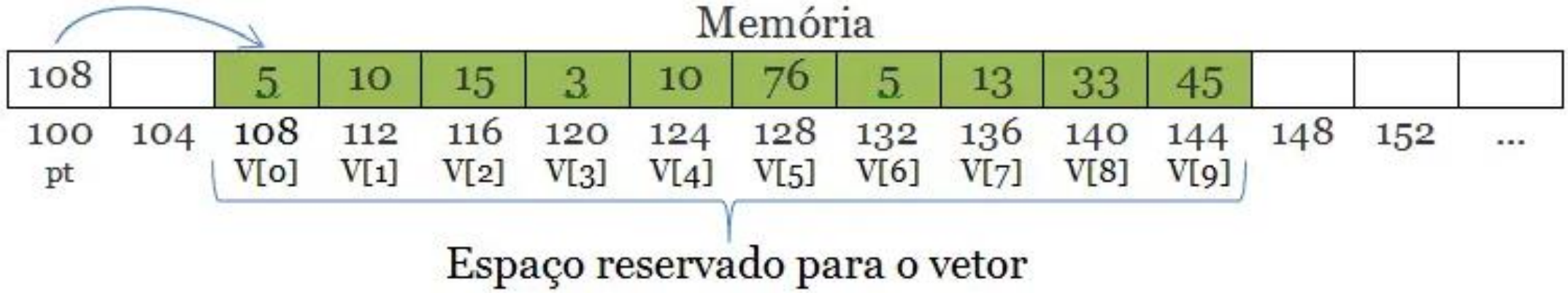
- Vetores são um conjunto de dados de mesmo tipo e que são armazenados em posições contíguas da memória [1].
- Considere, por exemplo, um vetor de inteiros com 10 elementos armazenados a partir do endereço 108.
- Um ponteiro é criado e passa a apontar para o primeiro elemento do vetor.
- Para atribuir o endereço de um vetor para um ponteiro basta utilizar o próprio nome do vetor

Ponteiros e Vetores

- Para atribuir o endereço de um vetor para um ponteiro basta utilizar o próprio nome do vetor

```
int v[] = {5, 10, 15, 3, 10, 76, 5, 13, 33, 45};
int * pt;

pt = v; //atribui o endereço do vetor
```



Ponteiros e Vetores

- Para obter o endereço de outro índice é necessário utilizar o operador ‘&’. Portanto, as duas atribuições mostradas abaixo são equivalentes.

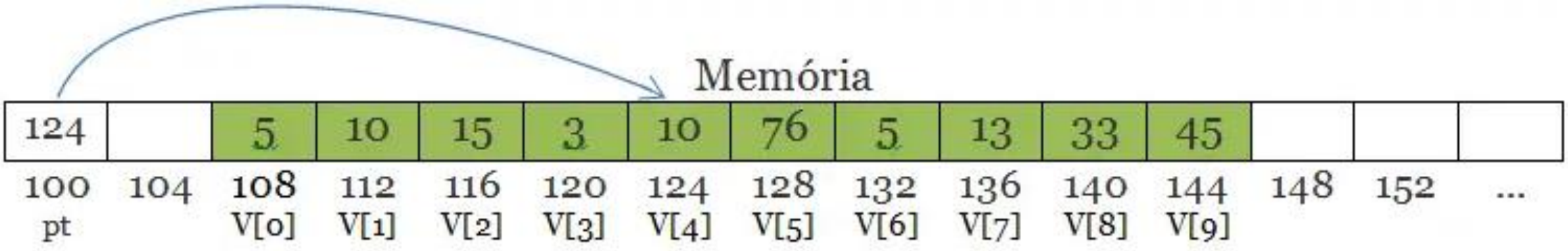
```
1  pt = v;
2  pt = &v[0];
```

- Logo, o endereço do quinto elemento pode ser obtido da seguinte forma:

```
1  pt = &v[4];
```

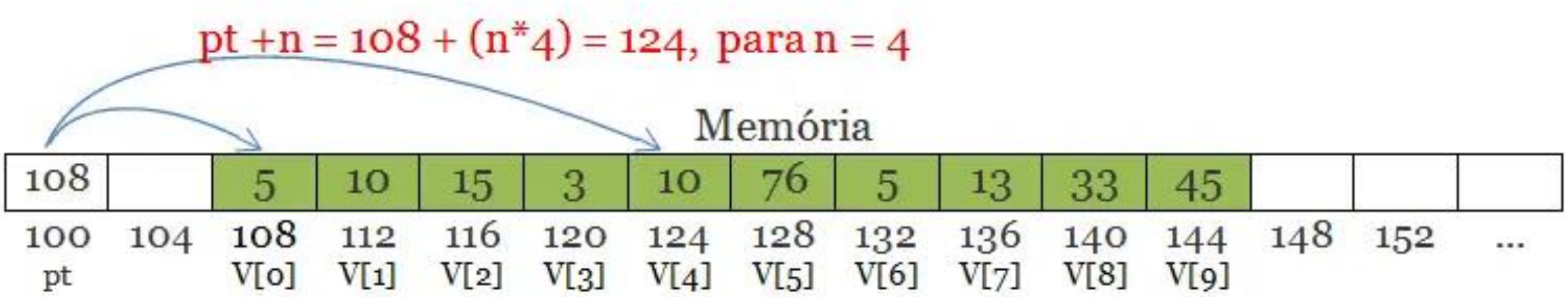
Ponteiros e Vetores

```
pt = &v[4];
```



Ponteiros e Vetores

- Se *pt* aponta para o endereço base do vetor, então '*V[n]*' é equivalente '**(pt + n)*'



Ponteiros e Vetores

- Para exibir os 10 elementos desse vetor

```
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 + i));
}
```

Ponteiros e Vetores

- Para exibir os 10 elementos desse vetor

```c
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 + i));
}
```

```
V[0] = 5
V[1] = 10
V[2] = 15
V[3] = 3
V[4] = 10
V[5] = 76
V[6] = 5
V[7] = 13
V[8] = 33
V[9] = 45
```

Ponteiros e Vetores

De modo geral:

- *(pt + i) é igual a V[i].
- (pt + i) é igual a &V[i].

Ponteiros e Vetores

- Quando declaramos uma matriz da seguinte forma, por exemplo:

```
int mat[20][30];
```

- O compilador C calcula o tamanho, em bytes necessário para armazenar esta matriz. Este tamanho é:

```
20 x 30 x 2 bytes
```

- O compilador então aloca este número de bytes em um espaço livre de memória;

Ponteiros e Vetores

- Consideremos o seguinte trecho de código:

```
char str[30], *p;
p = str;
```

- Neste caso p foi inicializado com o endereço do primeiro elemento da matriz st;

- Como faríamos para acessar o quinto elemento em str?

```
str[4];     ou    *(p + 4);
```

Ponteiros e Vetores

- Um dos usos mais importantes dos ponteiros: A varredura sequencial de uma matriz:

```
int main () {
char str[30], *p;
p = str;
 printf("Varredura sequencial com ponteiros");
printf("\n Digite um nome: ");
gets(str);
printf("\n O nome digitado foi: ");
while (*p)
printf("%c", *p++);
return 0;
}
```

Ponteiros e Vetores

- Um dos usos mais importantes dos ponteiros: A varredura sequencial de uma matriz:

```
int main () {
char str[30], *p;
p = str;
 printf("Varredura sequencial com ponteiros");
printf("\n Digite um nome: ");
gets(str);
printf("\n O nome digitado foi: ");
while (*p)
printf("%c", *p++);
return 0;
}
```

O ponteiro p aponta para a posição 0 (zero) da string str

O ponteiro p é incrementado e passa a apontar para próxima posição da string str

O ponteiro p é testado até atingir o delimitador /0

Ponteiros e Vetores

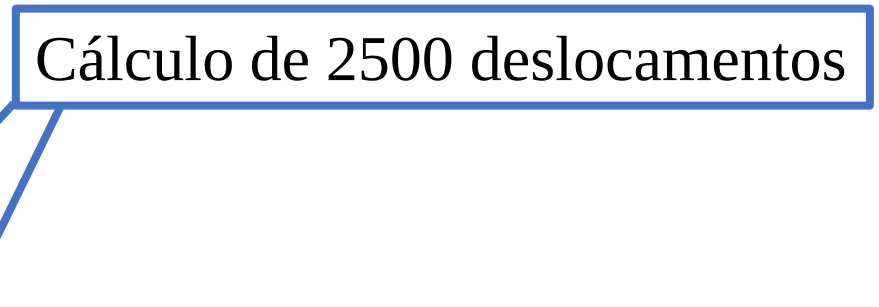
- Vejamos outro exemplo de varredura sequencial de uma matriz, onde se torna ainda mais evidente a eficácia dos ponteiros:

```
int main () {
float mat[50][50];
int i, j, count;
for (i=0; i<50; i++)
    for (j=0; j<50; j++)
        mat[i][j] = count++;
return 0;
}
```

Ponteiros e Vetores

- Vejamos outro exemplo de varredura sequencial de uma matriz, onde se torna ainda mais evidente a eficácia dos ponteiros:

```
int main () {
float mat[50][50];
int i, j, count;
for (i=0; i<50; i++)
    for (j=0; j<50; j++)
        mat[i][j] = count++;
return 0;
}
```



Ponteiros e Vetores

- Vejamos outro exemplo de varredura sequencial de uma matriz, onde se torna ainda mais evidente a eficácia dos ponteiros:

```
int main () {
float mat[50][50];
int i, j, count;
for (i=0; i<50; i++)
    for (j=0; j<50; j++)
        mat[i][j] = count++;
return 0;
}
```

Cálculo de 2500 deslocamentos

```
int main () {
float mat[50][50], *f, count;
for (count=0; count<2500; count++){
    *f = count;
     f++;
}return 0;
}
```

Ponteiros e Vetores

- Vejamos outro exemplo de varredura sequencial de uma matriz, onde se torna ainda mais evidente a eficácia dos ponteiros:

```c
int main () {
float mat[50][50];
int i, j, count;
for (i=0; i<50; i++)
    for (j=0; j<50; j++)
        mat[i][j] = count++;
return 0;
}
```

Cálculo de 2500 deslocamentos

```c
int main () {
float mat[50][50], *f, count;
for (count=0; count<2500; count++){
    *f = count;
    f++;
}return 0;
}
```

Apenas o incremento de ponteiro

Ponteiros e Vetores

- Faça um programa em C que leia uma matriz mat 2 x 3 de inteiros e encontre o menor elemento da matriz.

Ponteiros e Vetores

- Faça um programa em C que leia uma matriz mat 2 x 3 de inteiros e encontre o menor elemento da matriz.

```c
int main () {
int mat[2][3], count, var, menor=600000, *p;
p = mat[0];

printf("Este programa encontra o menor elemento de uma matriz\n");
for(count=0; count<6; count++, p++) {
    printf("Digite o elemento %d: ", count+1);
    scanf("%d", &var);
    *p = var;
        if ( *p <= menor)
        menor = *p;
}
printf("\nO menor valor da matriz eh: %d", menor);
return 0;
}
```

Ponteiros e Vetores

- Os ponteiros também podem ser declarados na forma de vetores.
- Defina um vetor de ponteiros com 4 elementos e mais quatros vetores de 3 elementos.

Ponteiros e Vetores

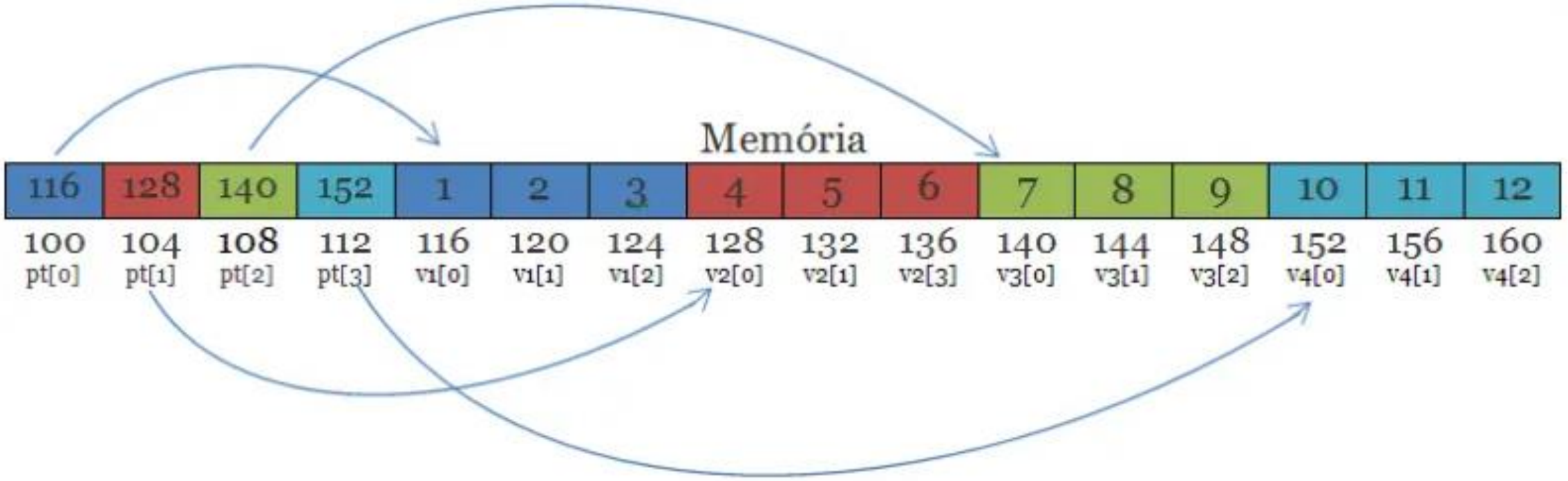
- Os ponteiros também podem ser declarados na forma de vetores.
- Defina um vetor de ponteiros com 4 elementos e mais quatros vetores de 3 elementos.

```
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]
```

Ponteiros e Vetores

- Defina um vetor de ponteiros com 4 elementos e mais quatros vetores de 3 elementos.



Ponteiros e Vetores

- 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.

•*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].

Ponteiros e Vetores

- *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]	V1
((pt+0))	1
((pt+0)+1))	2
((pt+0)+2))	3

pt[1]	V2
((pt+1)))	4
((pt+1)+1))	5
((pt+1)+2))	6

pt[2]	V3
((pt+2)))	7
((pt+2)+1))	8
((pt+2)+2))	9

pt[3]	V4
((pt+3))	10
((pt+3)+1))	11
((pt+3)+2))	12

Ponteiros e Vetores

- Ponteiros podem ser organizados em vetores como qualquer outro tipo de dado;
- A declaração de uma matriz de ponteiros int, de tamanho 10, é:

```
int *x[5];
```

- Para atribuir o endereço de uma variável inteira, chamada var, ao terceiro elemento da matriz de ponteiros, deve-se escrever:

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

- Para encontrar o valor de var, escreve-se:

```
*x[2];
```

Ponteiros

Funções e Alocação dinâmica de memoria.

- Sizeof
- Malloc;
- Calloc;
- Realloc;
- Free.

FIM!