Estrutura de Dados 1

Prof. Igor Calebe Zadi
igor.zadi@ifsp.edu.br

INSTITUTO FEDERAL
São Paulo
Campus Catanduva

I. Fundamentos de Estruturas de Dados

INSTITUTO FEDERAL
São Paulo
Campus Catanduva

I. Fundamentos de Estruturas de Dados

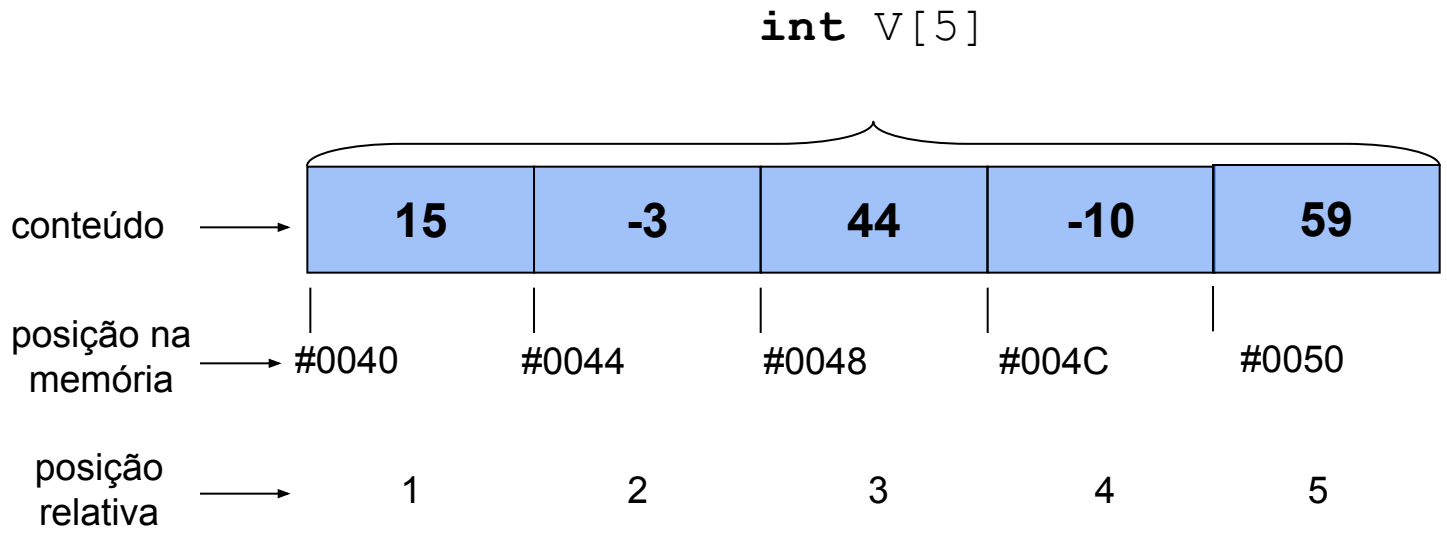
1. Definições
2. Classificação das Estruturas de Dados
3. Programação Orientada a Procedimentos
4. Tipos de dados primitivos
5. Cadeias de caracteres
6. Registros
7. Ponteiros

7. Ponteiros (alocação dinâmica)

INSTITUTO FEDERAL
São Paulo
Campus Catanduva

7. Ponteiros – alocação dinâmica

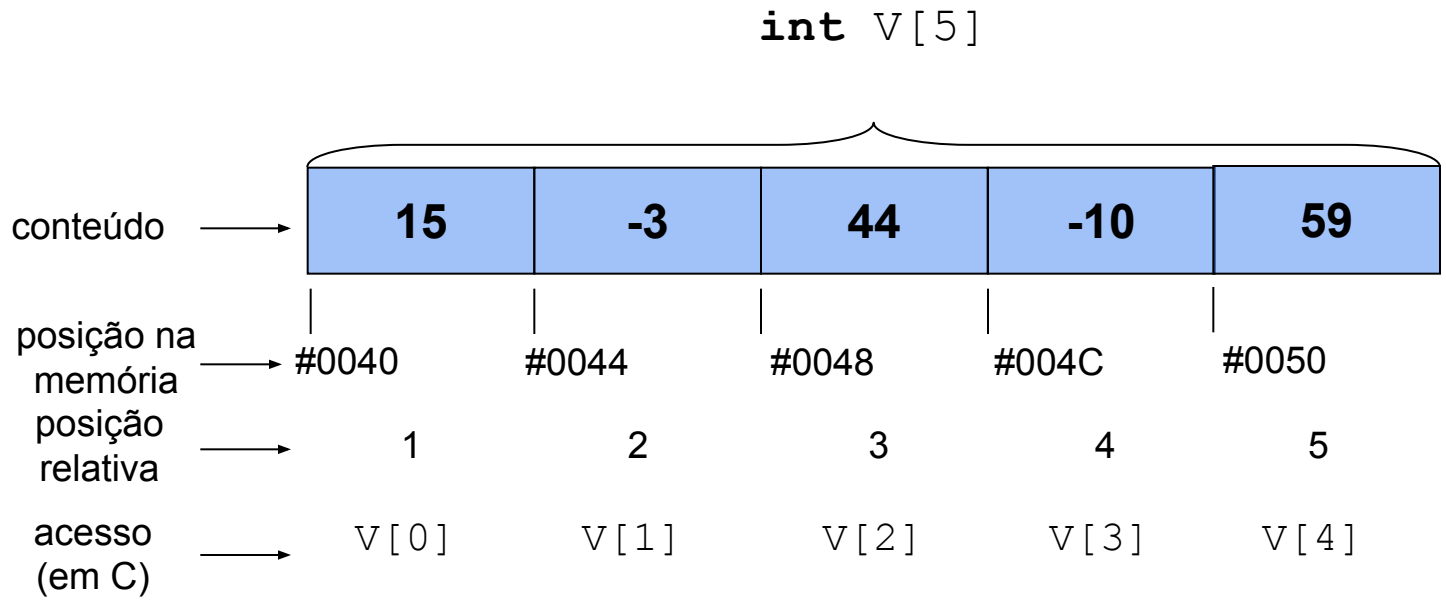
- Vetor (revisão)
  - Arranjo no qual as posições de memória são logicamente ocupadas em ordem sequencial crescente
    - necessita de apenas uma informação posicional



INSTITUTO FEDERAL
São Paulo
Campus Catanduva

7. Ponteiros – alocação dinâmica

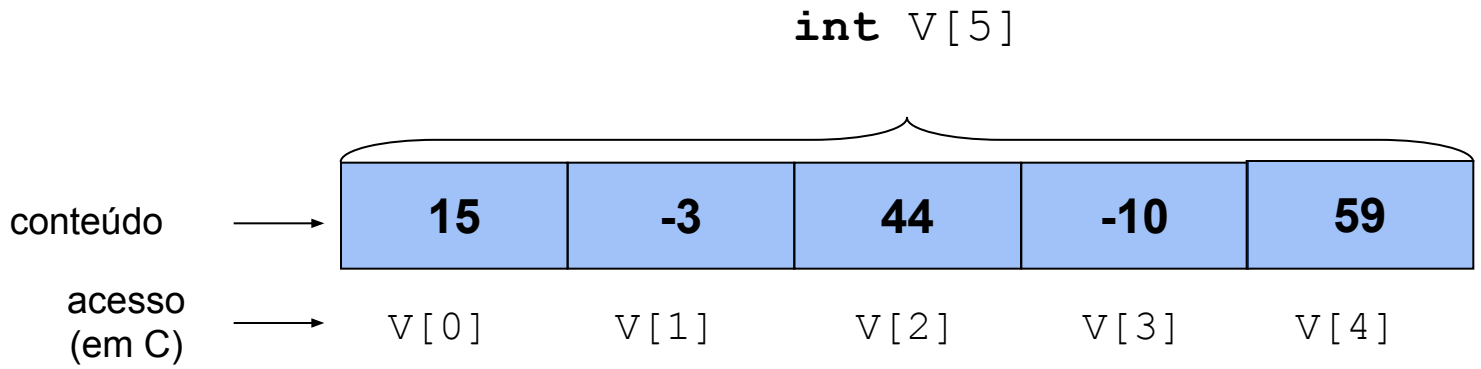
- Acesso a um elemento do vetor
  - nome do vetor + informação posicional
    - a informação posicional recebe o nome de **índice**
    - contagem do índice depende de cada linguagem de programação (linguagem **C**: começa do zero)



INSTITUTO FEDERAL
São Paulo
Campus Catanduva

7. Ponteiros – alocação dinâmica

- Índice (ou subscrito)
  - proporciona acesso direto a cada elemento do vetor (não é preciso acessar os elementos anteriores àquele desejado)
  - é um número inteiro ou uma expressão que resulta em um número inteiro
    - para consistência: 1 ≤ índice ≤ tamanho do vetor
  - o símbolo **[ ]** é um operador



INSTITUTO FEDERAL
São Paulo
Campus Catanduva

7. Ponteiros – alocação dinâmica

- Índice
  - permite que cada elemento do vetor receba valor individualmente, por atribuição ou leitura

```
...
V[i] = -8;
```

  - em outras palavras ...
    - cada elemento do vetor comporta-se como uma variável isolada e independente das demais
    - respeita-se apenas o tipo de dado (inteiro, real, etc.) declarado para o vetor

7. Ponteiros – alocação dinâmica

- Alocação de memória
  - **estática**: o tamanho do vetor é definido na própria declaração
    - não pode ser modificado
  - **dinâmica**: o tamanho do vetor pode ser modificado ao longo da execução do programa
    - não precisa ser explicitamente informado no código
  - ocupação efetiva da memória dependerá sempre do tipo de dado associado ao vetor

7. Ponteiros – alocação dinâmica

- Alocação estática
  - exemplos de declaração em linguagem C

```
int vet1[50];   // ocupa (50*sizeof(int)) posições
float vet2[50]; // ocupa (50*sizeof(float)) posições
int vet3[] = {1,5,9}; // 3*sizeof(int)) posições
```

7. Ponteiros – alocação dinâmica

- Alocação dinâmica
  - memória é alocada (e liberada) na **medida da necessidade**
  - linguagens possuem funções para alocação e liberação de memória

7. Ponteiros – alocação dinâmica

- Funções de alocação em **C**
  - malloc()
  - calloc()
  - realloc()
  - free()

7. Ponteiros – malloc()

- Sintaxe:

```
int *ptr = (int*) malloc(10 * sizeof(int));
```

  - Aloca espaço para 10 inteiros.
  - Retorna ponteiro para o primeiro endereço alocado.
  - Pode retornar NULL se a alocação falhar.

7. Ponteiros – calloc()

- Sintaxe:

```
int *ptr = (int*) calloc(10, sizeof(int));
```

- Semelhante ao malloc, porém inicializa a memória com zeros.

7. Ponteiros – realloc()

- Usado para redimensionar um bloco de memória alocado anteriormente.

- Sintaxe:

```
ptr = (int*) realloc(ptr, 20 * sizeof(int));
```

- Pode mover os dados para um novo endereço se necessário.

7. Ponteiros – free()

- Libera a memória alocada dinamicamente.
- Evita vazamento de memória.

- Sintaxe:

```
free(ptr);
ptr = NULL;
```

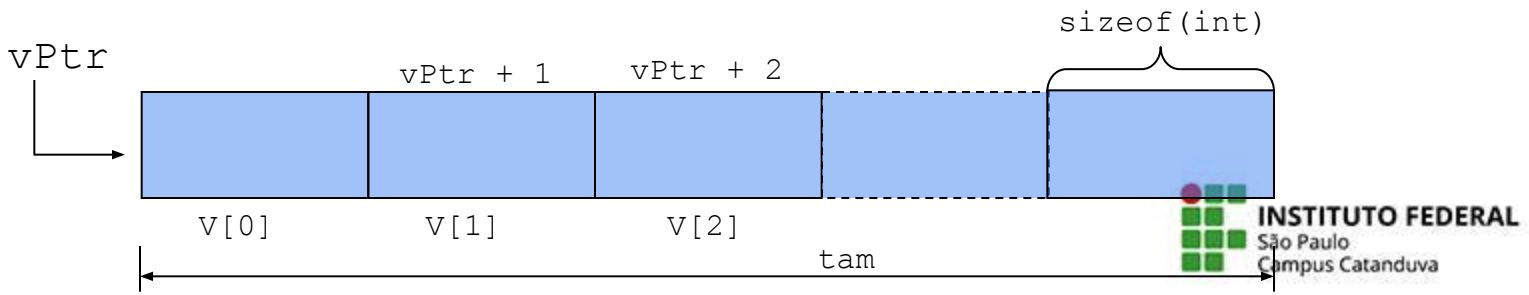
7. Ponteiros – alocação dinâmica

- Alocação dinâmica
  - importante, pois um mesmo programa pode ser reutilizado com diferentes instâncias de dados, **sem necessidade de recompilação**
  - ideia
    - alocar memória necessária ao problema
    - utilizar o respectivo ponteiro como se fosse um endereço para o vetor
    - aritmética de ponteiros: notação **ponteiro/deslocamento**
    - índices: notação **ponteiro/índice**

7. Ponteiros – alocação dinâmica

- Alocação dinâmica – exemplo em **C**

```
#include <stdlib.h>  /* biblioteca com funções de alocação */
int main ( ) {
   int * vPtr;    /* ponteiro que receberá endereço da área */
   int   tam;     /* qtd elementos do vetor - a ser obtido */
   ...
   /* obtém valor de 'tam' por teclado ou arquivo */
   ...
   /* aloca memória para o vetor */
   vPtr = (int *) calloc ( tam, sizeof(int) ); /* casting */
   if ( vPtr != NULL ) {
      ... /* usar *(vPtr+i) ou vPtr[i] */
      free ( vPtr );  /* após usar vPtr, libera a memória */
   }
}
```



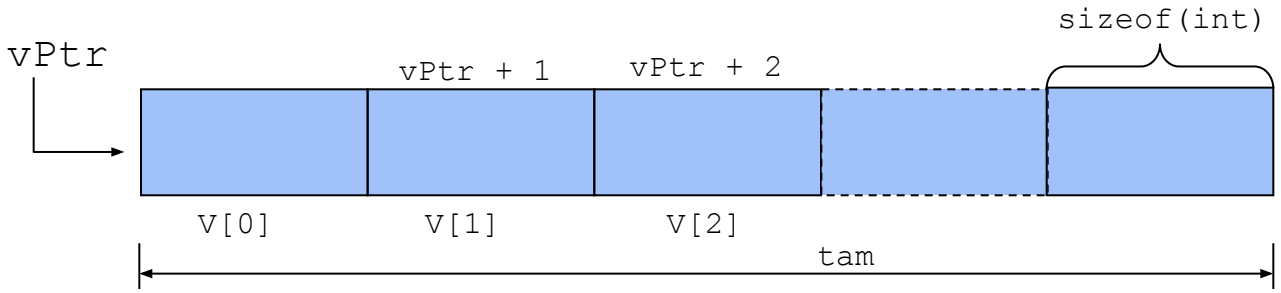
7. Ponteiros – alocação dinâmica

- Alocação dinâmica – exemplo em **C**

```
for(int i = 0; i < tam; i++){

        printf("%d",*(vPtr+i));

}
```



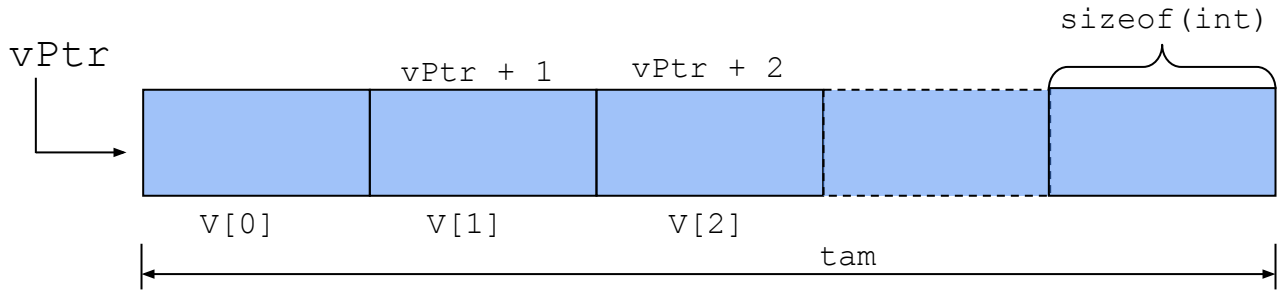
INSTITUTO FEDERAL
São Paulo
Campus Catanduva

7. Ponteiros – alocação dinâmica

- Alocação dinâmica – exemplo em **C**

```
int * xPtr; //ponteiro temporário (atenção!!!)
xPtr = vPtr; //recebe o endereço original do vetor

for(int i = 0; i < tam; i++){
  printf("xPtr = %d", *xPtr);
  xPtr++;//incremento do ponteiro
}
```



INSTITUTO FEDERAL
São Paulo
Campus Catanduva

7. Ponteiros

- Ponteiro
  - Sempre verificar se a alocação foi bem-sucedida.
  - Sempre liberar a memória após o uso.
  - Uso excessivo pode levar a fragmentação de memória.

Observações sobre o material eletrônico

- O material ficará disponível na pasta compartilhada que é acessada sob convite
- O material foi elaborado a partir de diversas fontes (livros, internet, colegas, alunos etc.)
- Alguns trechos podem ter sido inteiramente transcritos a partir dessas fontes
- Outros trechos são de autoria própria
- Esta observação deve estar presente em qualquer utilização do material fora do ambiente de aulas do IFSP - Catanduva

INSTITUTO FEDERAL
São Paulo
Campus Catanduva