# Estrutura de Dados e Algoritmos

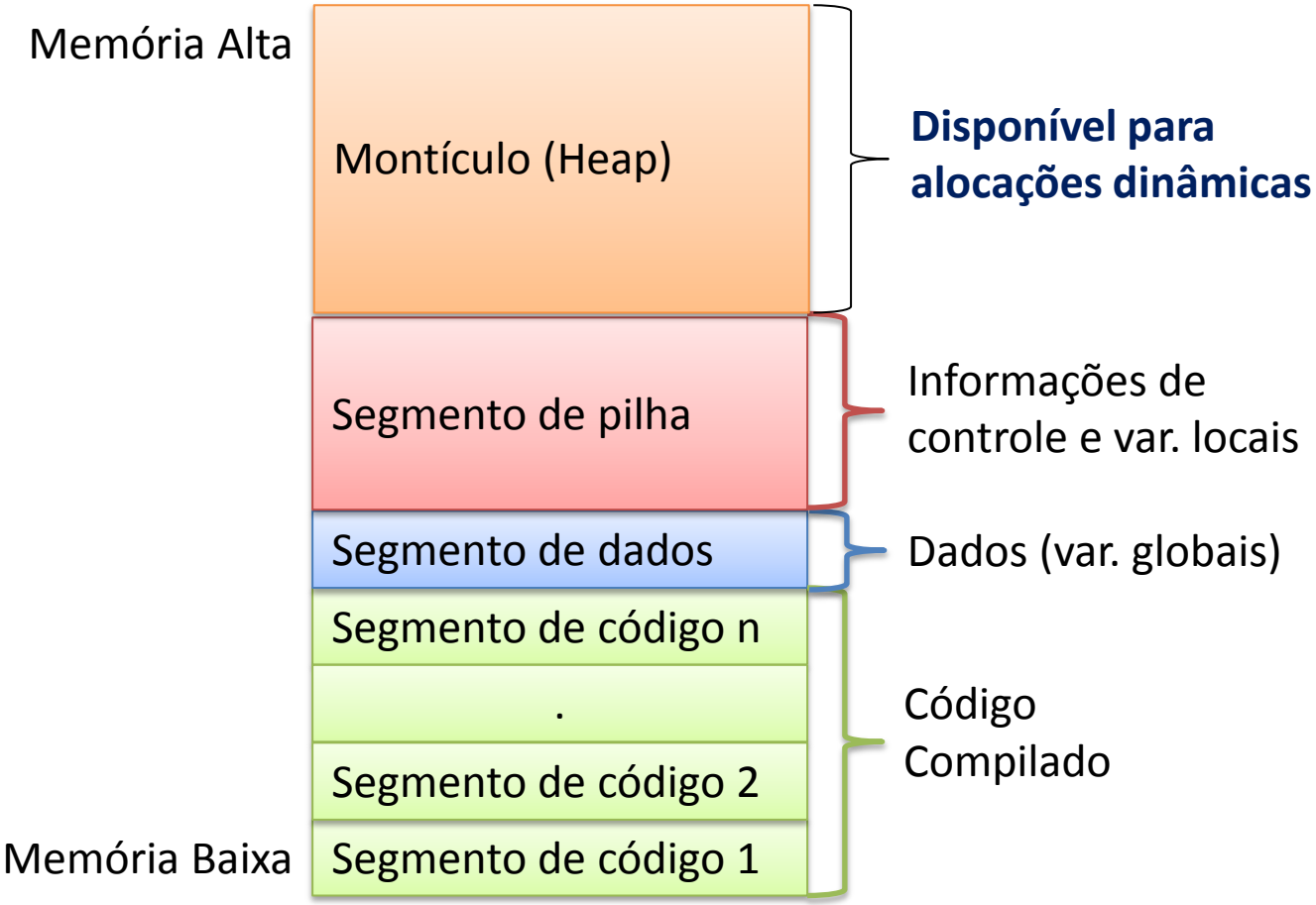
## Alocação Dinâmica de Memória

# Gerenciamento Dinâmico de Memória

- Necessário sempre que a quantidade de memória requerida só é sabida em tempo de execução:
  - Carregamento dos dados de um arquivo;
  - Operações com vetores/matrizes cujas dimensões dependem do contexto do problema;
  - …

# Gerenciamento Dinâmico de Memória

- Mapa de memória de um programa:



# Gerenciamento Dinâmico de Memória

- Vantagens:
  - Controle do uso de memória: uso sob demanda;
- Cuidados:
  - Gerenciar lixo;
- Implementação em C/C++:
  - Via ponteiros;
  - Funções/Operadores de alocação e desalocação;

# Vetores Dinâmicos em Linguagem C

- malloc:

  void* malloc (size_t qtde);

  *qtde: quantidade de bytes a serem alocados.*

```
tipo do dado *nome = (tipo do dado *) malloc(sizeof(tipo do dado) * tamanho);
```

- calloc:

  void* calloc (size_t qtde, size_t tam);

  *qtde: quantidade de elementos a serem alocados.*

  *tam: tamanho (em bytes) de cada elemento.*

```
tipo do dado * nome = (tipo do dado *) calloc (tamanho, sizeof(tipo do dado));
```

# Vetores Dinâmicos em Linguagem C

- free:

  void free(void* ptr);

  *ptr: ponteiro para a área de memória a ser desalocada.*

- *#include <stdlib.h>*
  - malloc, calloc
  - free

# Vetores Dinâmicos em Linguagem C

```
// Exemplo - C: malloc, calloc, free
#include <stdlib.h>     // Biblioteca de: malloc, calloc, free

int main ()
{
  int *Vet1, *Vet2;                          //Declaração dos ponteiros
  Vet1 = (int*) malloc (100*sizeof(int));    //Alocação de memória
  Vet2 = (int*) calloc (100, sizeof(int));   //Alocação de memória
  free (Vet1);                               //Desalocação de memória
  free (Vet2);                               //Desalocação de memória
  return 0;
}
```

# Vetores Dinâmicos em Linguagem C++

- new:

  tipoD *nome = new tipoD[qtde];

  *qtde: quantidade de elementos do tipo **tipoD** a serem alocados.*

```
tipoD *nome = new tipoD[tamanho];
```

- delete[]:

  void delete[];

```
delete[] Vetor;
```

*Vetor: ponteiro alocado pelo comando new*

- **new e delete são implementados como operadores** da própria linguagem C++. Isto significa que não há necessidade de incluir nenhuma biblioteca ao código.

# Vetores Dinâmicos em Linguagem C++

```
// Exemplo – C++: new e delete[]
int main ()
{
  int pind, ptam;
  float *pVetor;                    //Declaração dos ponteiros
  printf (“Digite a qtde de elementos: ");
  scanf ("%d",ptam);
  pVetor = new float[ptam];         //Alocação de memória
  for(pind=0;pind<ptam;pind++)
     pVetor[pind] = ((float)pind)/ptam;
  for(pind=0;pind<ptam;pind++)
     printf (“%f“,pVetor[pind]);
  delete[] pVetor;                  //Desalocação de memória
  return 0;
}
```

# Vetores Dinâmicos em Linguagem C

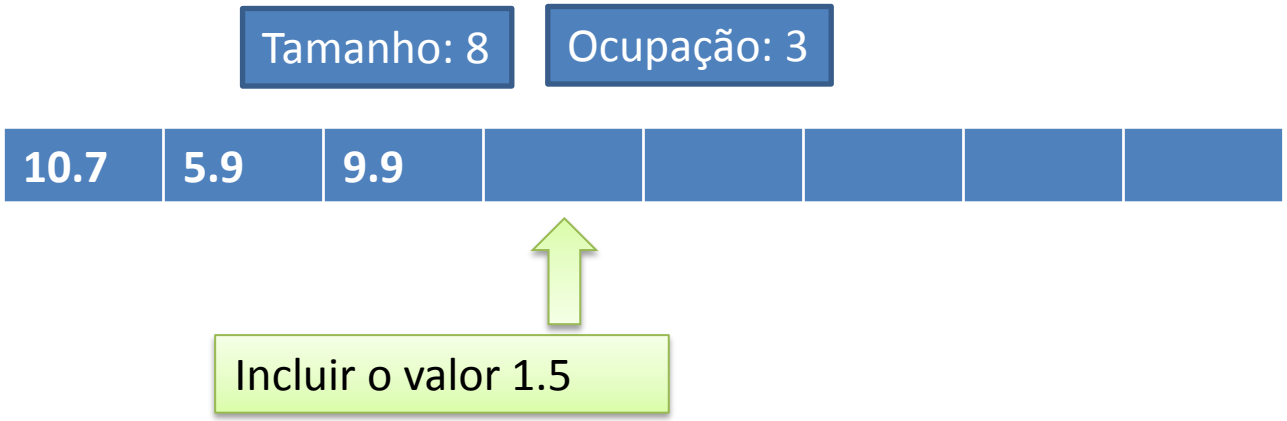
- Exercícios...
  - Exercício 1;
  - Exercício 2.

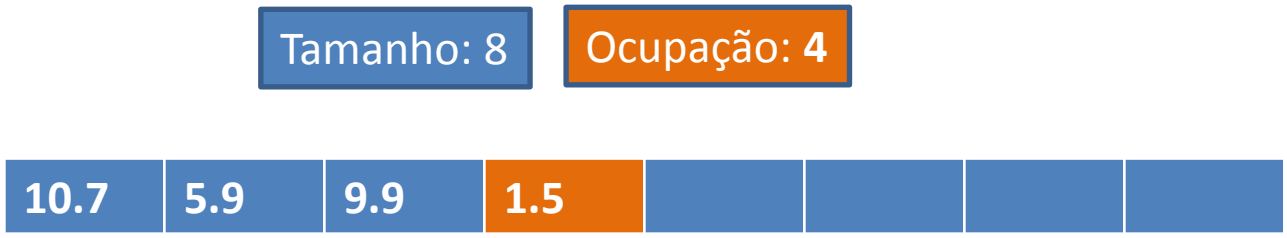
# Vetores Dinâmicos

- Um vetor, seja alocado estaticamente, seja alocado dinamicamente, não guarda informações sobre o seu tamanho;
- Sempre que criar funções que recebem vetores como parâmetros de entrada, lembre-se de passar o tamanho do vetor também.

# Vetores Desordenados

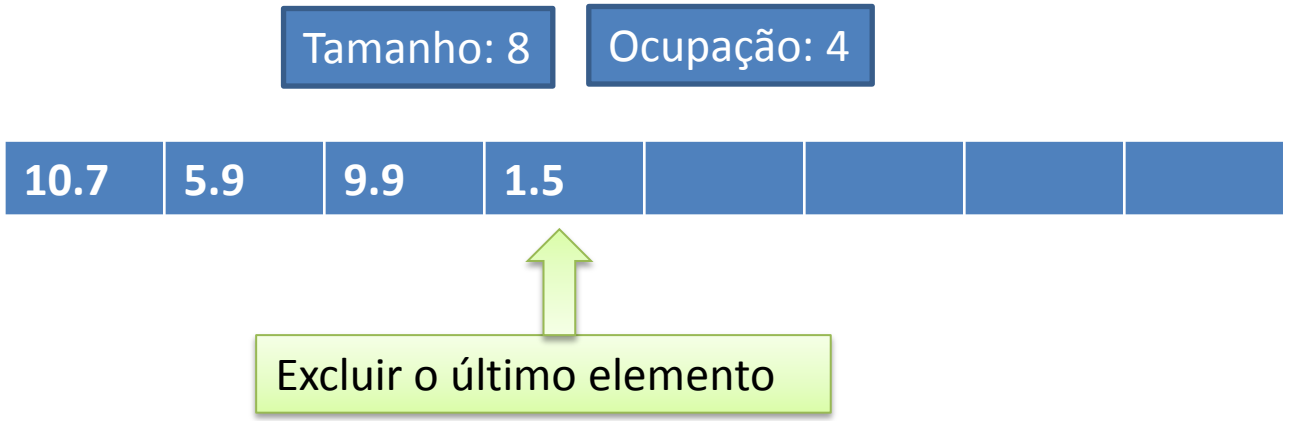
- Inclusão:

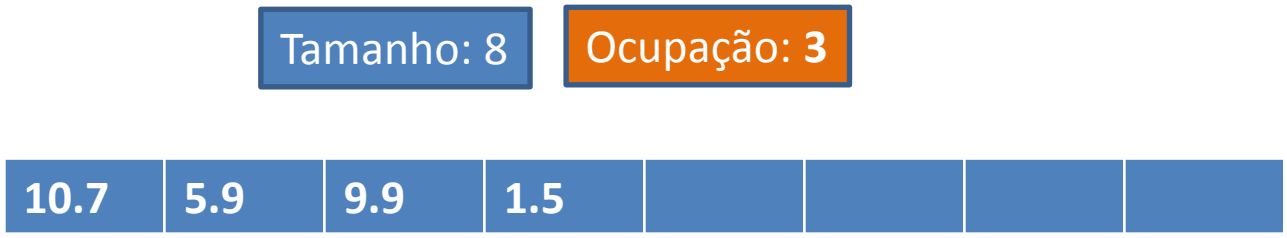


# Vetores Desordenados

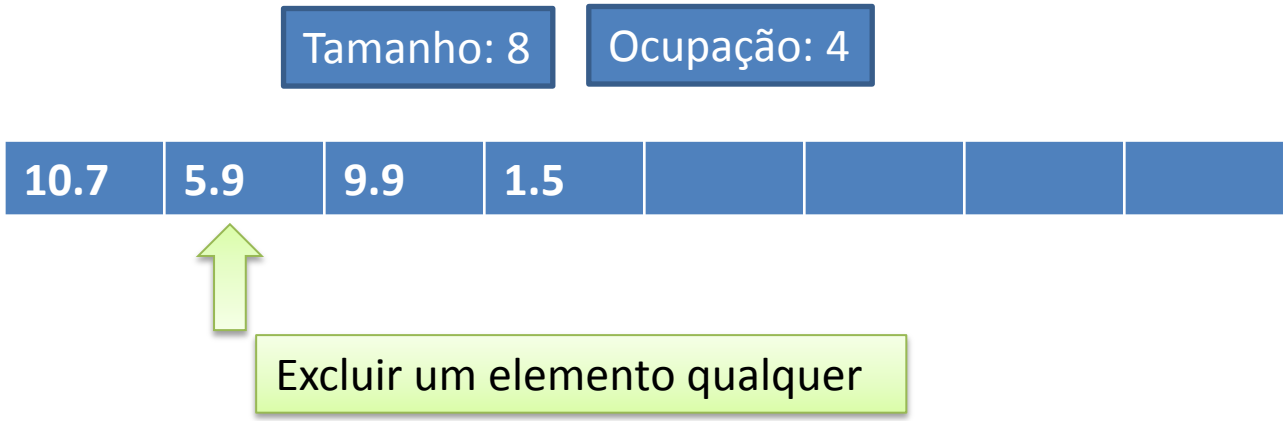
- Exclusão:

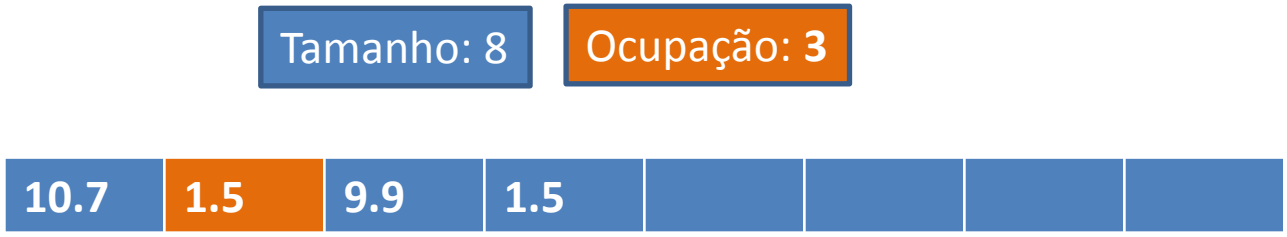


# Vetores Desordenados

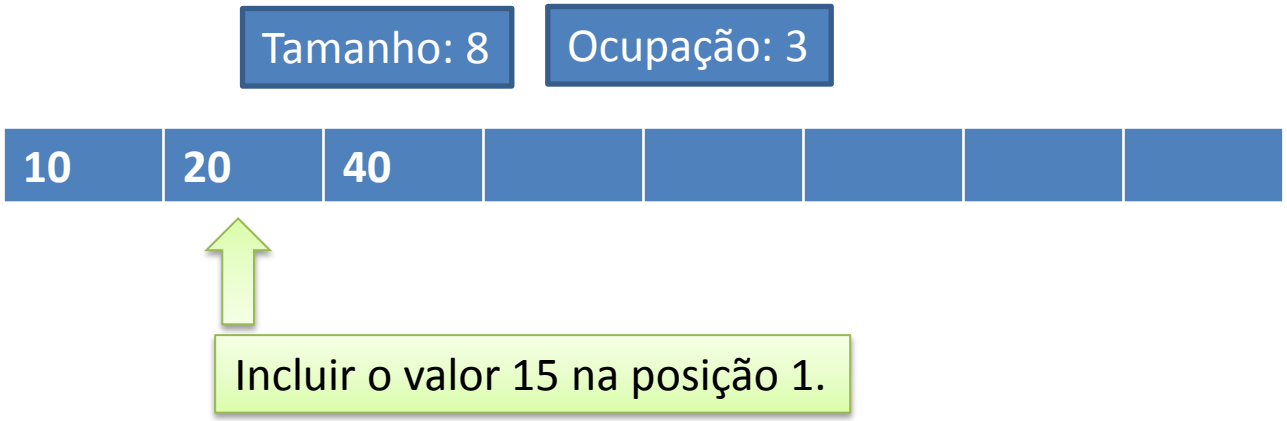
- Exclusão:

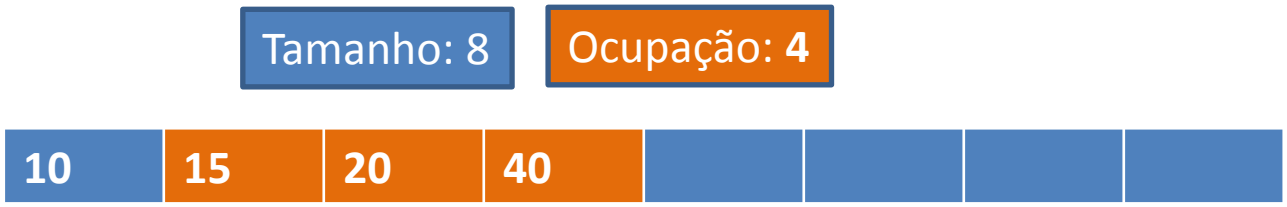


# Vetores Ordenados

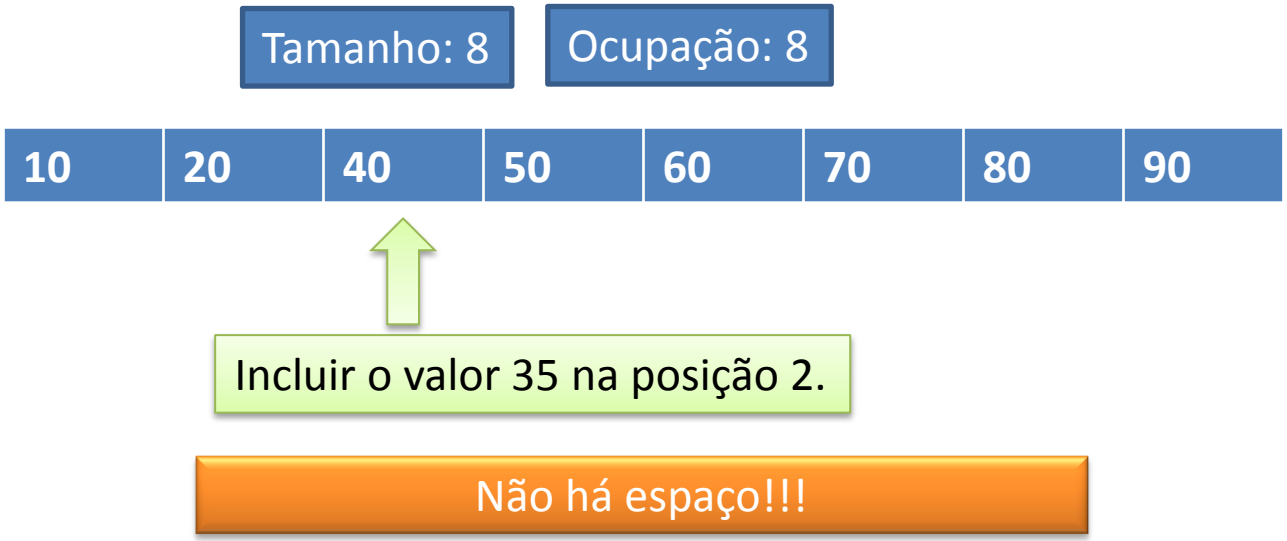
- Inclusão:



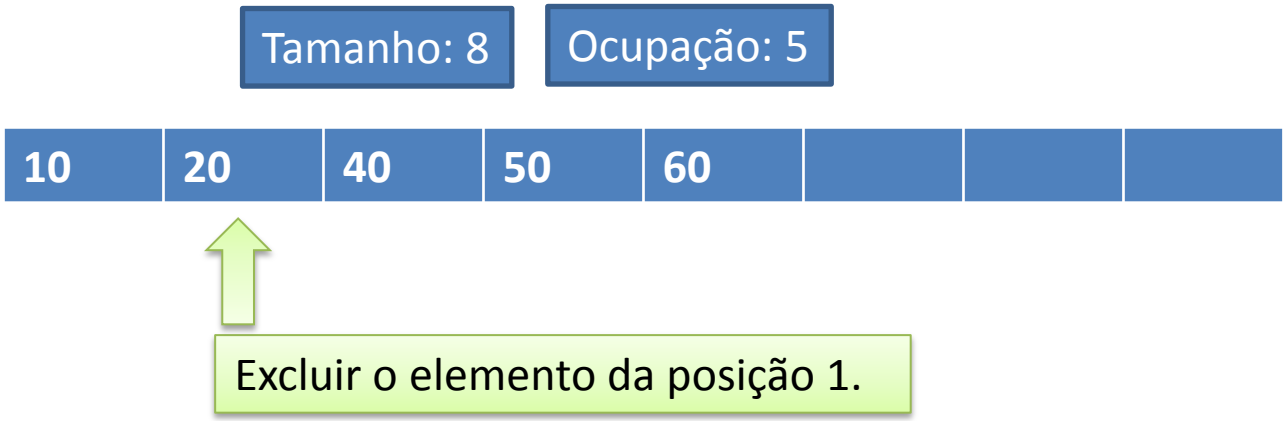
# Vetores Ordenados

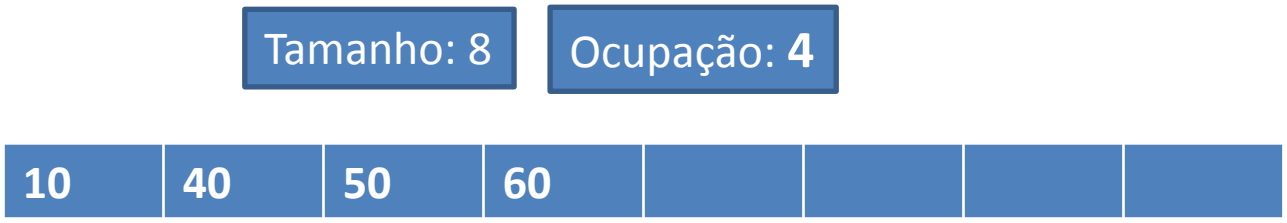
- Inclusão:



# Vetores Ordenados

- Exclusão:



# Prática...

- No ambiente virtual (Moodle):
  - Prática Vetores Dinâmicos.pdf
  - Arquivo de dados: notas.txt