Algoritmos e Estruturas de Dados

(Aula 5 - Criando uma biblioteca de manipulação de vetores)

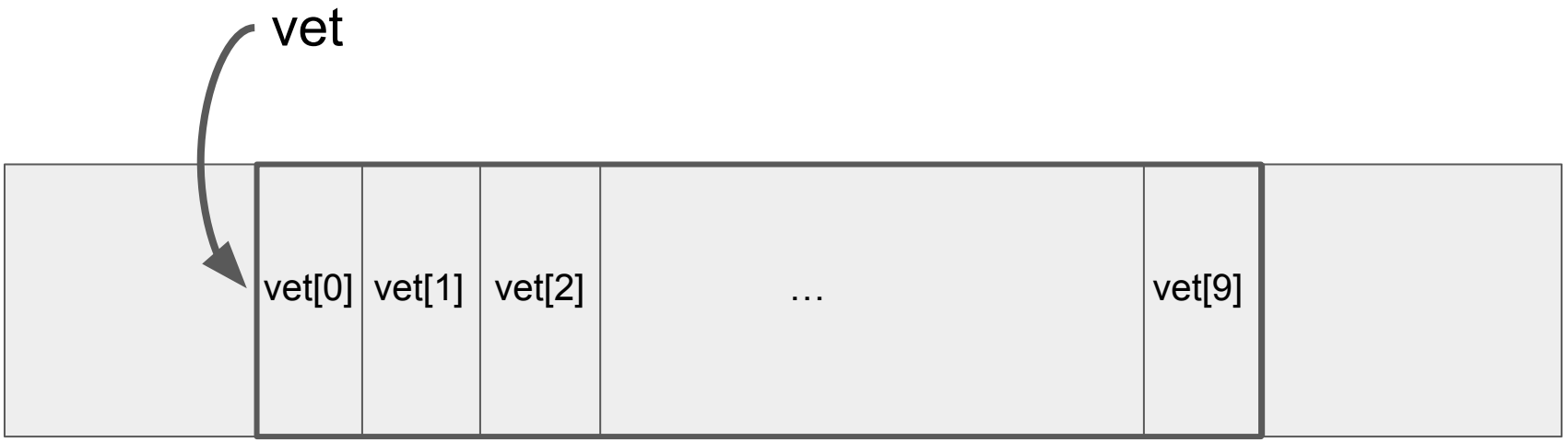
Prof. Me. Diogo Tavares da Silva
contato: *diogotavares@unibarretos.com.br*

FaculdadeBarretos
www.faculdadebarretos.com.br

Contextualização

- aula anterior:
  - Vimos como funciona um vetor ou matriz estática em C/C++
    - Uma variável que referencia o início do bloco de dados
    - tamanho definido em tempo de compilação
    - alocação de memória contígua

```
int vet[10];
```



Estruturas Sequenciais

- Projetar algoritmos de operação sobre esta estrutura de dados, ou seja, sobre a estrutura de vetor:
  - **Criação**
    - criar a estrutura de vetor
  - **Percurso**
    - percorrer a estrutura
  - **Inserção**
    - inserir dados em uma determinada posição
  - **Busca**
    - Buscar uma informação na estrutura
  - **Remoção**
    - Remover uma informação da estrutura

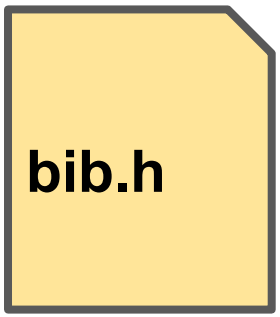
Estruturas Sequenciais

- Em todas as estruturas que estudaremos aqui:
  - aplicar conceitos de reúso de software e modularização
    - vamos criar uma biblioteca que poderemos importar para processar esse tipo de estrutura em qualquer programa que escrevermos..
  - Como?
    - arquivos de cabeçalho (*headers*)
    - “biblioteca.h”

Criação da biblioteca

- Para criar uma biblioteca:

Arquivo em que estarão implementadas as funções, variáveis, estruturas que importaremos



Arquivo de *header* contendo as interfaces (protótipos) das funções e estruturas desenvolvidas em **bib.cpp**

Arquivo principal que irá importa nossa biblioteca sempre que necessário (**#include "bib.h"**)

Estruturas Sequenciais

- Ok! então vamos projetar nossa biblioteca para vetor
  - Pontos de projeto:
    - tamanho da estrutura
      - usar alocação estática ou dinâmica?
      - usar e controlar uma variável inteira **t** para manter o tamanho atualizado

Criação

- Projetar função que cria o vetor
  - Necessário compreender a relação entre ponteiros, vetores e alocação dinâmica

Ponteiros e Vetores

- Vetores e matrizes são na verdade ponteiros
  - apontam pra primeira posição da coleção de dados

```
int vet[3];
for (int i = 0; i< 3; i++){
    cin >> *(vet+i); //equivale a vet[i]
}
cout << vet << endl; //end da posição 0
cout << vet[0] << endl; //valor da posição 0
cout << *(vet) << endl; //equivalente
```

Alocação dinâmica de vetores

- Ao invés de declararmos um valor com um tamanho preestabelecido em tempo de compilação
  - Podemos declarar um ponteiro
  - definir um tamanho em tempo de execução
  - Alocar a quantidade necessária de memória usando o comando **new**
  - Podemos também desalocá-lo usando o comando **delete**
  - **Vantagens:**
    - **aloca-se apenas a quantidade de memória desejada**

Alocação dinâmica

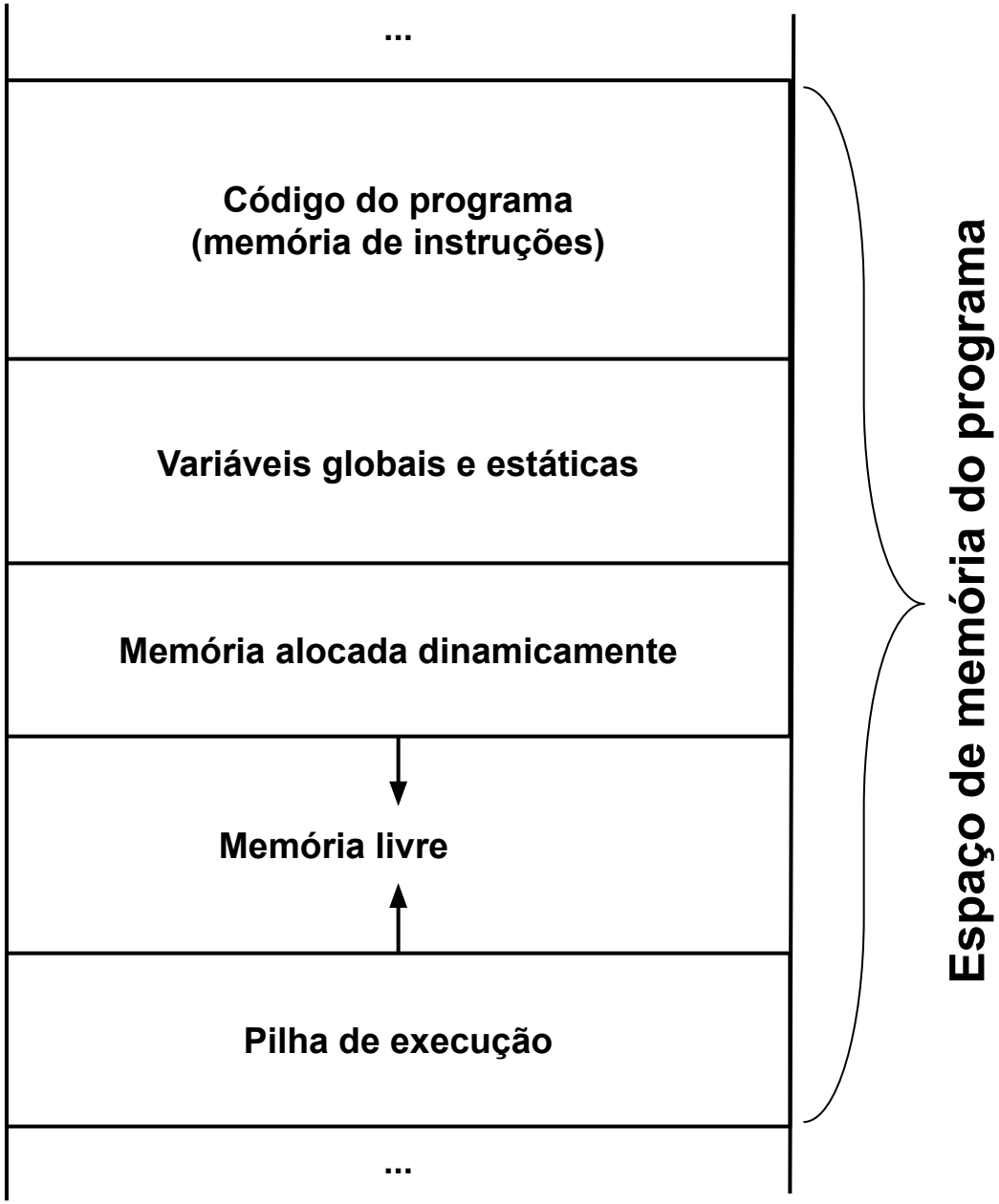
- Exemplo:
- ...

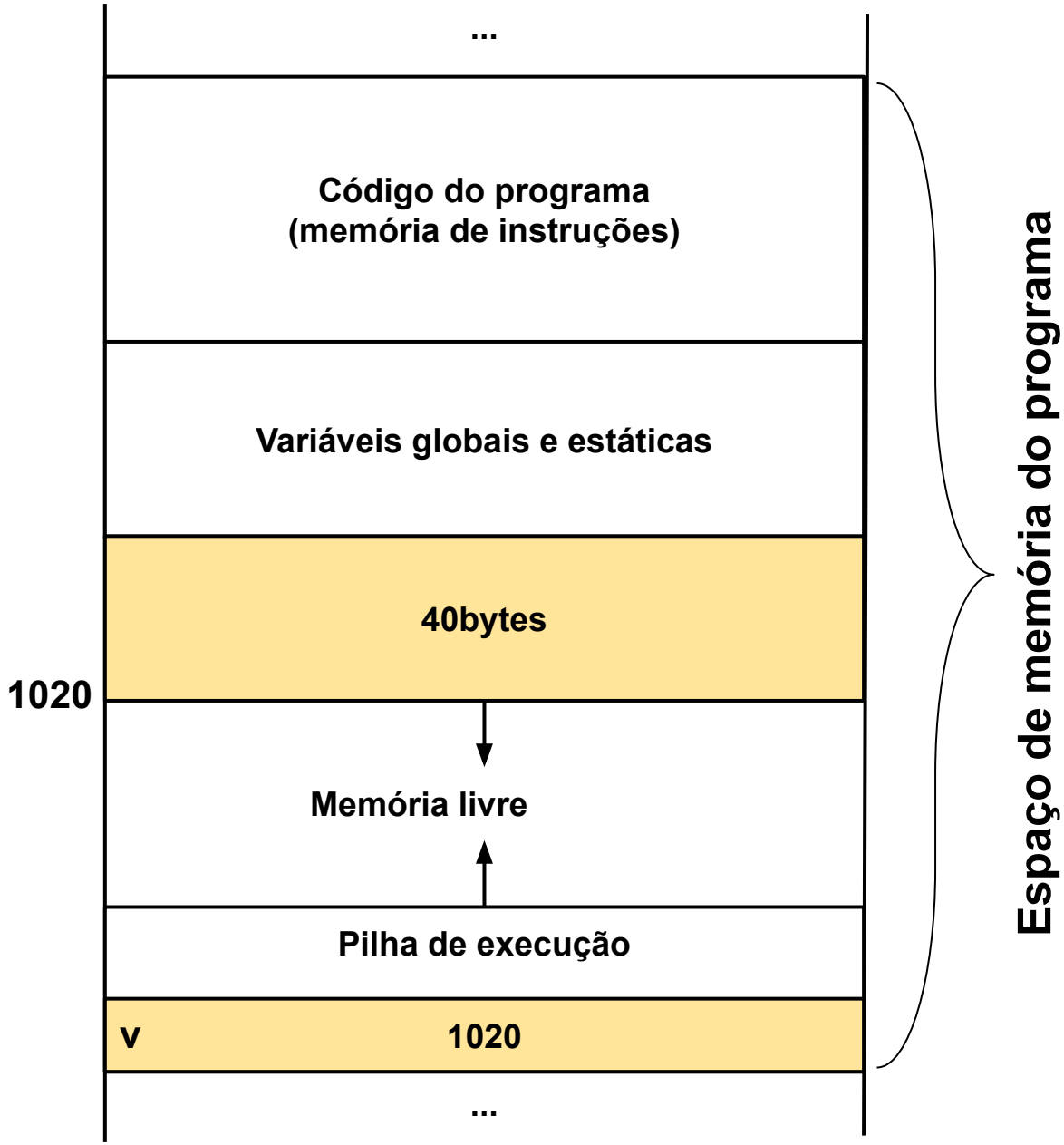
```
int *v;

v = new int[10]; //em C++

//ou

v = malloc(sizeof(int)*10); //em C puro
```





Alocação dinâmica

- exemplo:

```
int *v;
int tam;
cout << "Tamanho do vetor: ";
cin >> tam;

v = new int[tam]; //cria um vetor de tamanho "tam"

for(int i = 0; i< tam; i++){
    cin >> v[i];
}
for(int i = 0; i< tam; i++){
    cout << v[i] << endl;
}
```

Alocação dinâmica

- usando função:

```
int* criaVetor(int tam){
    int *v;
    v = new int[tam];
    return v;
}
```

```
int *nums, t;
t = 7;
nums = criaVetor(t);

    for(int i = 0; i< t; i++){
    cin >> nums[i];
}
for(int i = 0; i< t; i++){
    cout << nums[i] << endl;
}
```

Alocação dinâmica

- **Exercício:**
  - Pesquise e projete uma função ***criaMatriz()*** que instancie uma matriz dinâmica segundo os parâmetros ***nLinhas***, ***nColunas*** e ***valorIni*** e retorne-a para um ponteiro que irá manipular a matriz.
  - **DICA:** A alocação de uma matriz dinâmica é análoga ao processo de alocação de um vetor, sendo que uma matriz bidimensional é um ponteiro de ponteiros (vetor de vetores)

Percurso

- Projetar função
  - parâmetros:
    - vetor a ser manipulado, tamanho do vetor
  - exemplo:
    - imprimir

Percurso

```
void printVetor(int t, int vet[]){
    for (int i=0; i< t; i++){
        cout << "Valor da pos [" << i << "]: ";
        cout << vet[i] << endl;
    }
}
```

Percurso

- **exercícios:**
- Projetar funções
  - maior, menor, soma, média

Inserção

- **Problema:**
  - Inserir um elemento no vetor
    - onde?
      - início
      - final
      - meio
        - qual critério?
  - **considerações:**
    - aceitar elementos repetidos

Inserção

- **Problema:**
  - desafios:
    - vetor é uma estrutura alocada sequencialmente
      - inserir no início ou no meio
        - implica em “empurrar” os valores subsequentes uma posição à frente…
      - Inserir no fim
        - insere e aumenta o tamanho do vetor

Inserção

- **Inserir no final**
  - **projeto da função**
    - **parâmetros:**
      - elemento a ser inserido, tamanho do vetor e vetor
    - **retorno:**
      - tamanho do vetor atualizado
      - e o vetor?
        - o vetor é passado por referência (ponteiro) e atualiza automaticamente

Inserção

- **Inserir no final**

```
int insereFim(int elem, int t, int vet[]){
    cout << "Inserindo " << elem << " na posição " << t << endl;
    vet[t] = elem;
    t++;
    return t;
}
```

*Custo de tempo e operações desta função?

Inserção

- **Inserir no meio**
  - sempre segundo algum critério
  - critério:
    - inserir ordenadamente (ordem crescente)

Inserção

- **Inserir no meio**
  - Ordem crescente
    - situações
      - vetor vazio
        - insere
      - elemento a ser inserido é o menor do vetor
        - insere início
      - elemento a ser inserido é o maior do vetor
        - insere fim
      - elemento no meio
        - percorrer vetor enquanto elem é maior que vet[i]
        - inserir e deslocar

```
int insereOrd(int elem, int t, int vet[]){
    cout << "inserindo " << elem << "..." <<endl << "t = " << t << endl;
    int pos=-1; //pos que vamos inserir
    //sit 1: vetor vazio
    if (t == 0){
        vet[0] = elem;
    }else{
        //sit 2 e 4: elem no inicio ou meio
        for (int i = 0; i< t; i++){
            if (elem < vet[i]){
                cout << elem << " < " << vet[i] << " ..." << endl;
                pos = i; //posição a inserir
                //percorre de trás pra frente
                //pra não precisar de auxiliar
                for (int j = t; j > pos; j--){
                    vet[j] = vet[j-1];
                }
                vet[pos] = elem;
                break;
            }

        }
        //sit 3: elem no final
        if (pos == -1){
            vet[t] = elem;
        }
    }
    t++; //atualiza o tamnho
    return t;
}
```

Inserção

- **exercícios**
  - função para inserir no início
  - inserir no meio fazendo as trocas de valores ao percorrer de frente pra trás com variável auxiliar

Busca

- **problema:**
  - buscar um elemento dentro do conjunto retornando seu índice

Busca

- **duas abordagens:**
  - **busca sequencial:**
    - se reduz ao problema de percurso
      - visitar o vetor inteiro até encontrar o valor deseja
    - **desafios**
      - tamanho do vetor influencia no custo da busca
  - **busca binária:**
    - dividir para conquistar
      - vai dividindo o espaço de busca pela metade dos elementos restantes
    - **desafios**
      - vetor precisa estar previamente ordenado

Busca

- **busca sequencial:**
  - **projeto da função:**
    - **parâmetros:**
      - elemento a ser buscado, tamanho do vetor e vetor
    - **retorno:**
      - índice do elemento buscado
        - (ou -1 se elemento não encontrado)

Busca

- **busca sequencial:**

```
int buscaSeq(int elem, int t, int vet[]){
    int pos = -1;

    for (int i = 0; i< t; i++){
        if (elem == vet[i])
            pos = i;
            break;
    }

    return pos;
}
```

Busca

- **busca sequencial:**

```
int buscaSeq(int elem, int t, int vet[]){
    int pos = -1;

    for (int i = 0; i< t; i++){
        if (elem == vet[i])
            pos = i;
            break;
    }

    return pos;
}
```

Remoção

- **Problema:**
  - Remover um determinado elemento do vetor
    - **de onde?**
      - final
        - apenas atualizar o tamanho
      - início ou meio
        - deslocar todas os elementos subsequentes para a posição anterior
    - **considerações:**
      - remover a partir de uma posição previamente buscada
      - remover todas as ocorrências

Remoção

- **Remover um elemento específico**
  - **1º passo:**
    - buscar o elemento
      - se reduz ao problema da busca
  - **2º passo:**
    - remover elemento buscado

Remoção

- **Remover elemento a partir de um índice buscado**
  - **projeto da função:**
    - **parâmetros:**
      - índice do elemento a ser removido, tamanho do vetor, vetor.
    - **retorno:**
      - tamanho do vetor atualizado
    - **Considerações**
      - lembrar antes de testar o índice retornado para chegar se o elemento buscado existe no vetor

Remoção

- **Remover elemento a partir de um índice buscado**

```
int removeElem(int pos, int t, int vet[]){
    for (int i = pos+1; i<t; i++){
        vet[i-1] = vet[i];
    }
    t--;
    return t;
}
```

Remoção

- **Exercícios**
  - Criar funções:
    - Remover do início
    - Remover do fim
    - Remover todos elementos repetidos
      - dica: para cada posição percorrer o restante buscando a repetição e removendo-as

Trabalho (Para entrega)

- Construa uma biblioteca (vetor.c e vetor.h) de manipulação de vetores que possua as funções de:
  - Criação do vetor
  - Percurso (impressão)
  - Inserção no início
  - Inserção Ordenada
  - Busca Sequencial
  - Busca Binária
  - Remoção no Final

Trabalho (Para entrega)

- **Para busca binária, investigue como funciona o algoritmo e o implemente nos moldes de nossa biblioteca:**
  - Função
    - **int buscaBinariaVetor(int elem, int t, int vet[])**
      - Busca o elemento *elem* no vetor *vet* de tamanho *t,* utilizando o método de busca binária
      - retorna o índice do elemento buscado ou -1, caso não encontrado.

Trabalho (Para entrega)

- Façam em grupos de até 3 alunos.
- Entregar para diogotavares@unibarretos.com.br
  - assunto: [AED-T1]
  - Nome de todos os integrantes no corpo da mensagem!
  - Anexar arquivos!
  - Não serão toleradas cópias de trabalhos!