Estrutura de Dados

Aula 2

Listas Lineares Sequenciais

prof Leticia Winkler

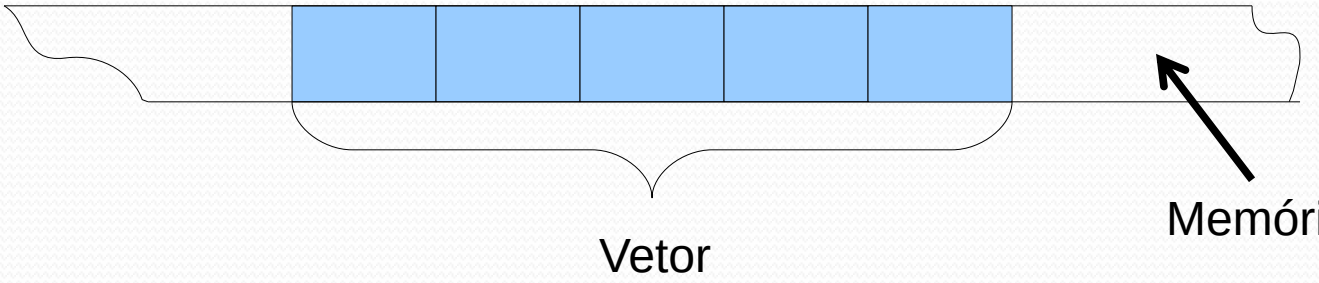
Conteúdo

- Revisão de Vetor
  - Definição
  - Declaração
  - Acesso dos Elementos
  - Inicialização dos Elementos
  - Exemplo
- Vetor e Função
- Lista Linear
  - Operações com Listas
  - Tipos de Listas Lineares
    - Pilha
    - Fila
    - Deque
- Lista Linear Sequencial
  - Codificação em C/C++ das Operações com listas lineares sequenciais

Revisão de Vetor

Definição de Vetor

- São estruturas homogêneas que possibilitam o armazenamento de um conjunto de valores em um espaço contiguo da memória
  - as posições de memória onde estão armazenados os elementos de um mesmo vetor estão em seqüência
  - estrutura de dados homogênea porque todos os seu elementos devem ser do mesmo tipo de dado.



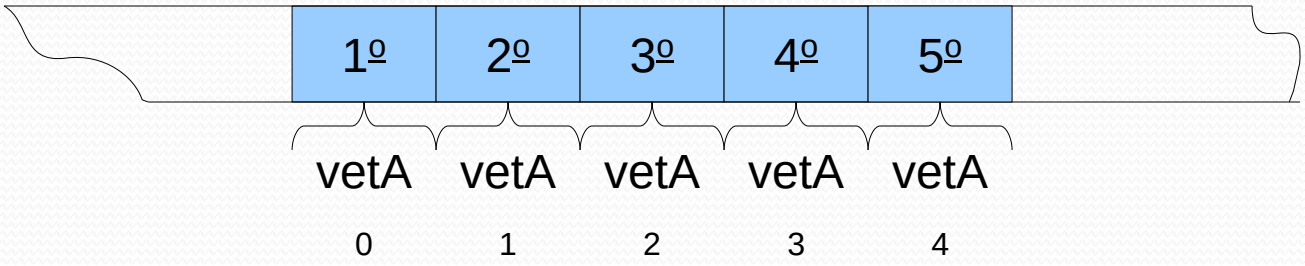
Declaração de um Vetor

tipo nome_do_vetor[tamanho];

- *tipo* indica o tipo do dado dos valores a serem armazenados no vetor
- *nome_do_vetor* indica o nome da variável vetor
- *tamanho* indica a quantidade de elemento que o vetor pode armazenar.
- Exemplo :int vetA[5];

Elementos de um Vetor

- Cada elemento do vetor é acessado por um índice
  - representa a posição relativa do elemento em relação ao vetor.
- Em C/C++ o primeiro elemento tem índice o (zero).



- Acesso em C++:

  vetA[**3**] é o **quarto** elemento do vetor

Inicialização dos Elementos

- Um vetor pode ser inicializado na declaração:

```
int vetA[5] = {15, 2, 35, 70};
```

- Através de um expressão de atribuição:

```
vetA[1] = 23;
```

Exemplo #1

```
#include <iostream>
using namespace std;

int main (){
        int vet[10];

        cout << "Digite os elementos do vetor:\n"
        for (int i=0; i<10; i++){
                cout << "Elemento da posição " << i << ": ";
                cin >> vet[i];
        }

        cout << "Conteúdo do vetor – ordem inversa: ";
        for (int i=9; i>=0; i--){
                cout << "Elemento da posição " << i << ": " <<vet[i]<< endl;
        }
    return 0;
}
```

Exemplo #2

```
#include <iostream>
using namespace std;

int main() {
    int v[3], i;
    cout << "Entrando com dados para compor v : " << endl;
    for (i = 0; i < 3; i++) {
        cout << “Digite um valor : “;
        cin >> v[i];
    }
    cout << "Dobrando os valores de v ... " << endl;
    for (i = 0; i < 3; i++)
      v[i] = 2 * v[i];

    cout << "Imprimindo v após sua alteracao : " << endl;
    for (i = 0; i < 3; i++)
        cout << v[i] << endl;
    return 0;
}
```

Vetor e Função

```
#include <iostream>
#include <cstdlib>
using namespace std;
void lerDados(int x[], int n) {
   for (int i = 0; i < n; i++)
      cin >> x[i];
}
void dobrarDados(int x[], int n)   {
   for (int i = 0; i < n; i++)
      x[i] = x[i] * 2;
}
void imprimirDados(int x[], int n)
   for (int i = 0; i < n; i++)
```

```
int main() {
    int v[3], w[4], z[6];
    cout << "Entrando com dados para compor v : " <<
endl;
    lerDados(v, 3);
    cout << "Entrando com dados para compor w : " <<
endl;
    lerDados(w, 4);
    cout << "Entrando com dados para compor z : " <<
endl;
    lerDados(z,6);
    dobrarDados(v,3);
    dobrarDados(w,4);
    dobrarDados(z,6);
    cout << "Imprimindo v após sua alteracao : " <<
endl;
    imprimirDados(v,3);
    cout << "Imprimindo w após sua alteracao : " <<
endl;
    imprimirDados(w,4);
    cout << "Imprimindo z após sua alteracao : " <<
endl;
    imprimirDados(z,6);
    system("pause");
    return 0;
}
```

Passagem de parâmetros por referência e por valor

Lista Linear

- É uma estrutura de dados na qual os elementos estão organizados de maneira seqüencial.
- É formada por um conjunto de dados afins (de um mesmo tipo).
- O elemento é chamado de nó ou nodo.
- Preserva a relação de ordem entre seus elementos.
- Não necessáriamente os elementos estão fisicamente em ordem na memória
  - Quando os elementos estão fisicamente em ordem – lista linear sequencial (ou contigua)
  - Quando não estão fisicamente em ordem – lista linear encadeada

Operações com Listas

- Criação de uma lista
- Inicialização
- Remoção de uma lista
- Inserção de um elemento na lista
- Remoção de um elemento da lista
- Acesso de um elemento da lista
- Alteração de um elemento da lista
- Combinação de duas ou mais listas
- Classificação da lista
- Cópia da lista
- Localizar um elemento através de uma informação

Tipos de Listas Lineares

- De acordo com as operação que são permitidas serem realizadas, as listas lineares se classificam em:
  - pilhas
  - filas
  - deques

Pilhas

- É uma lista linear na qual o primeiro elemento a entrar é o último elemento a sair.
- Também é chamada de LIFO (“LAST IN FIRST OUT”), o último elemento que entrou, é o primeiro a sair.
- Possui apenas uma entrada, chamada de topo, a partir da qual os dados entram e saem dela.
- Exemplos de pilhas são:
  - pilha de pratos,
  - pilha de livros,
  - pilha de cartas de um baralho,
  - etc.

Fila

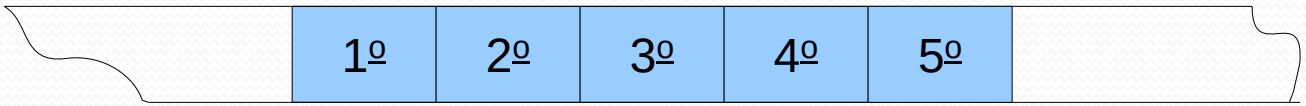
- É uma lista linear na qual o primeiro elemento a entrar é o primeiro elemento a sair.
- Também é chamada de FIFO (“FIRST IN FIRST OUT”) - O primeiro elemento que entrar será o primeiro a sair.
- Os elementos ”entram por trás e saem pela frente”.
- Exemplos de filas são:
  - Fila de caixa de banco,
  - Fila de vagões de trem,
  - Sala de espera de um médico
  - etc.

Deque

- Deque (double-ended queue)
- É uma lista linear na qual os elementos entram e saem tanto pela frente quanto por trás.
- Pode ser considerada uma generalização da fila.
- Exemplo:
  - Carregamento e descarregamento de containers por um navio num porto

Lista Linear Sequencial

- Os nós além de estarem em uma sequência lógica, estão também fisicamente em sequência.
- Os elementos são acomodados em um **vetor**.
  - A alocação de memória é estática (em tempo de compilação)
  - A forma de armazenamento na memória é contíguo ou sequencial.
    - Os dados são armazenados em endereços vizinhos de memória.
- Usa-se este tipo de lista quando se tem em mente um tamanho pré-definido, que não vá precisar redimensionar em tempo de execução.



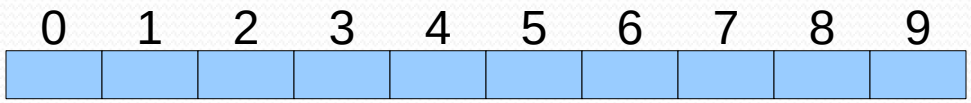
Listas Lineares Sequenciais

Codificação das Operações

Criar

- Numa Lista Linear Sequencial a criação da lista se dá com a declaração do vetor que irá conter os elementos da lista

```
int v[10];
int n;
```



  - V – representa o vetor
  - n – quantidade de elementos que a lista possui

Inicializar

- Inicializar significa preparar a lista para inserção dos elementos
- **n** indica quantos elementos a lista possui – inicialmente nenhum (0 elementos)
  - n, numa lista implementada através de um vetor, também pode ser visto como a próxima posição para inserção

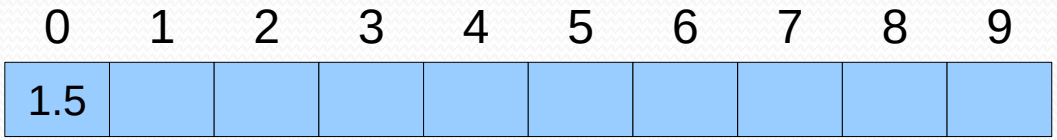
Inserir

- Considerando que a lista não precisa estar ordenada e pode ter elementos repetidos.
- Inserir um valor em uma lista não ordenada consiste em:
  - adicionar um valor na próxima posição disponível do vetor (inicialmente a posição disponível é o) e
  - ajustar a quantidade de dados, desde que a lista não esteja cheia.
- Parâmetros entrada:
  - A lista – ou seja, referência ao vetor;
  - Valor a ser inserido ou adicionado;
  - Quantidade de elementos na lista; e
  - Tamanho máximo do vetor.
- Saída (retorno da função): Não há

Função para Inserção

```
void inserir(float v[], float valor, int &n, int tamanho) {
      if (tamanho == n)
         cout << "ERRO: Lista cheia.";
      else {
          v[n] = valor;
          n++;
      }
}
```

Supondo que foi digitado o valor 1.5 para inserção



- Chamada na main:

```
...
cout << "Valor para inserção : ";
cin >> valor;
inserir(v, valor, n, 40);
...
```

Passagem de parâmetros por valor e por referência

Percorrer

- Consiste em mostrar o conteúdo da lista, imprimindo na tela os valores de seus elementos
- Parâmetros entrada:
  - A lista – ou seja, referência ao vetor
  - Quantidade de elementos existentes na lista.
- Saída (retorno da função) : Não há

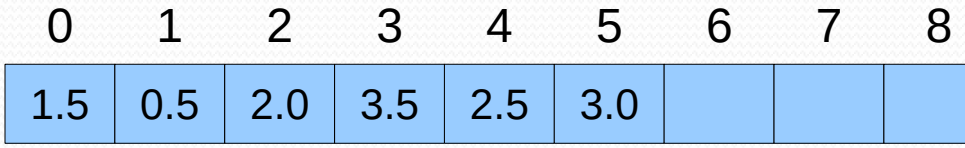
Função para Percorrer a Lista

```
void percorrer(float v[], int n) {
if (n == 0)
cout << "ERRO: Lista vazia.";
else {
for (int i = 0; i < n; i++)
cout << v[i] << '\t' ;
}
cout << endl;
}
```

- Chamada na main:

```
...
percorrer(v,n);
...
```

- Supondo a lista:



- Neste caso o valor de n é 6
- Será apresentado na tela:

  1.5 0.5 2.0 3.5 2.5 3.0

Buscar ou Pesquisar Sequencialmente

- Operação que procura, elemento por elemento, em sequencia, um valor em um vetor, que pode estar em ordem ou não.
- Consiste em procurar um valor, componente a componente, sequencialmente, retornando:
  - índice do valor (sucesso na busca – encontrou o elemento); ou
  - -1 (fracasso na busca – elemento não encontrado).
- Parâmetros entrada:
  - A lista – ou seja, referência ao vetor;
  - Valor a ser procurado; e
  - Quantidade de elementos existentes na lista.
- Saída (retorno da função):
  - índice do valor – caso seja encontrado; ou
  - -1 – caso o valor não seja encontrado
    - índice não válido para informar que não achou

Função para Buscar Sequencialmente

```
int buscarSequencial(float v[], float valor, int n) {
    if (n == 0) {
       cout << "ERRO: Lista vazia - ";
       return -1;
    }
    else {
       for (int i = 0; i < n; i++) {
            if (v[i] == valor) {
               return i;  // achou  – retorna o índice do dado
            }
       }
       return -1; // não achou  – retorna um índice impossível
    }
}
```

- Chamada na main:

```
cout << "Valor para busca ? ";
cin >> valor;
posicao = buscarSequencial(v,valor,n);
if (posicao >= 0)
   cout << "Elemento encontrado na posicao = " << posicao
     << endl;
else
   cout << "Elemento não encontrado " << endl;
```

- Supondo a lista:

0	1	2	3	4	5	6	7	8
1.5	0.5	2.0	3.5	2.5	3.0			

- Neste caso o valor de n é 6
- Supondo que deseja-se buscar o valor 3.5
- Será retornado da função *buscarSequencial* o valor 3
- Será apresentado na tela:

  Elemento encontrado na posicao = 3

Remover

- Considerando-se que não há ordem alguma na disposição dos dados da lista.
- Consiste em retirar da lista um elemento previamente escolhido.
  - Os dados do vetor serão ajustados, assim como a quantidade.
  - A remoção só poderá ocorrer, se a lista não estiver vazia e o elemento possa ser encontrado.
- Parâmetros:
  - A lista – ou seja, referência ao vetor;
  - Valor a ser removido da lista; e
  - Quantidade de elementos existentes no vetor.
- Saída (retorno da função): Não há

```
void remover(float v[], float valor, int &n) {
      if (n == 0) { // Testa se a lista está vazia
        cout << "ERRO : lista vazia." << endl;
        return;     // Abandona a função
      }
      int posicao = buscarSequencial(v,valor,n);
      if (posicao == -1) {
         cout << "ERRO : valor não encontrado." << endl;
         return;
      }
     // copia o último para a posição do valor a ser
removido
     v[posicao] = v[n-1];
     n--;  // ajusta a quantidade de elementos da lista
}
```

- Chamada na main:

```
cout << "Valor a ser removido? ";
cin >> valor;
remover(v,valor,n);
```

- Supondo a lista:

0	1	2	3	4	5	6	7	8
1.5	0.5	2.0	3.5	2.5	3.0			

- Supondo que deseja-se remover o valor 3.5
- Dentro da função remover, será retornado o valor 3 da função buscarSequencial

0	1	2	3	4	5	6	7	8
1.5	0.5	2.0	3.0	2.5	3.0			

- O valor de n passa a ser 5
- Observe que o 3.0 não é apagado, mas fica como lixo (desprezado)

Exercício

- Faça um programa que crie uma lista de inteiros distintos, através de sucessivas inserções e depois a apresente na saída padrão.
  - Implemente funções para :
    - Inserir um valor, sem que haja repetição. Para isto, será preciso fazer uma busca seqüencial e inserir o valor passado apenas se ele não existir na lista.
    - Percorrer a lista, imprimindo-a na saída padrão.
  - Protótipos :
    - void inserirSemRepetir(int [], int , int , int);
      - Parâmetros: Vetor de dados, elemento a ser inserido, quantidade de dados existentes no vetor e a quantidade máxima alocada para o vetor.
      - Obs.: A função buscarSequencial deverá ser chamada dentro da inserirSemRepetir e consequentemente, implementada no programa.
    - void percorrer(int [], int);
      - Parâmetros: Vetor de dados, quantidade de dados existentes no vetor