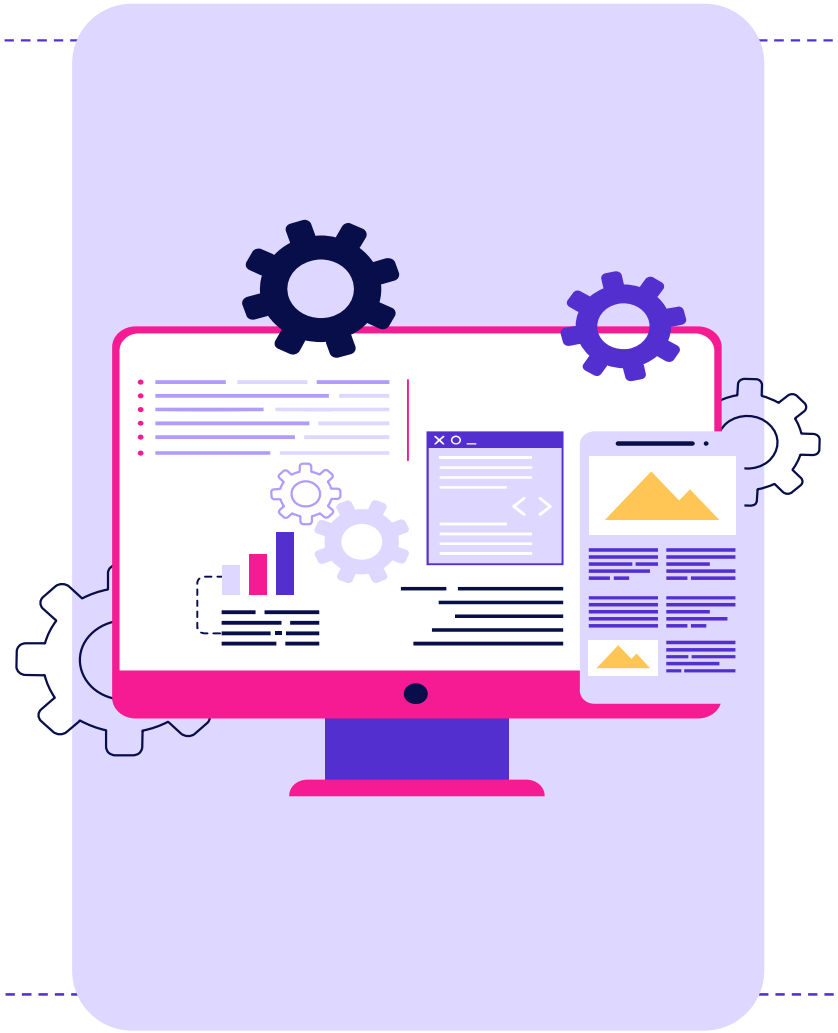
Estrutura de Dados 1

LISTA ESTÁTICA



Profª Juliana Franciscani

Roteiro

01 O que é

02 TADs

03 Condições (pré/pós)

04 Código

05 Exercícios

LISTA ESTÁTICA

Vantagens:

- Utiliza array/vetor na implementação
- Acesso rápido e direto aos elementos
- Tempo constante para acessar um elemento
- Facilidade em alterar informações
- Não há necessidade de compreender ponteiros ou referências.

Desvantagens:

- Limitações quanto ao tamanho de memória;
- Necessário definir o tamanho do vetor;
- Dificuldade para inserir e remover um elemento entre outros dois
- Custo computacional maior;
- Alocação de memória exagerada.

Principais Operações em Lista - TADs

- Criar lista;
- Apagar lista;
- Inserir item;
- Acessar item (dado uma chave);
- Remover item (dado uma chave);
- Contar número de itens;
- Verificar se a lista está vazia;
- Verificar se a lista está cheia;
- Copiar lista;
- Imprimir lista.

Condições para cada Operação:

Criar Lista

Pré-Condição
Existir espaço na memória

Pós-Condição:
Iniciar a estrutura de dados

Apagar Lista

Pré-Condição: Não há

Pós-Condição:
Remover a lista da memória

Inserir item

Pré-Condição:
Lista não está cheia

Pós-Condição:
Inserir item e retornar *true* se deu certo, e *false* se deu errado

Remover item (dado chave)

Pré-Condição: Não há

Pós-Condição:
Remover item da lista (chave), retorna *true* se a operação realizada, false caso contrário

Acessar Item (dado chave)

Pré-Condição: Não há

Pós-Condição:
Recuperar o item, retorna item se a operação realizada, **nulo** caso contrário

Verificar se Lista Vazia

Pré-Condição: Não há

Pós-Condição:
Retorna *true* se lista vazia e *false* se não estiver

Verificar se Lista Cheia

Pré-Condição: Não há

Pós-Condição:
Retorna *true* se lista cheia e *false* caso contrário

Imprimir Lista

Pré-Condição: não há

Pós-Condição:
Imprimir lista na tela/arquivo

Contar número de itens

Pré-Condição: Não há

Pós-Condição:
Retornar o número de itens na lista

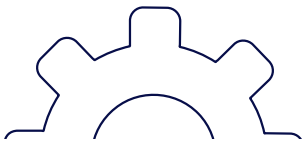
Copiar Lista

Pré-Condição
Lista existente, verificar Criar Lista()

Pós-Condição:
Fazer a cópia para a nova lista

Lista Estática

- Os itens são armazenados em um vetor de tamanho suficiente para armazenar a lista.
- Deve-se criar um marcador para indicar o último elemento.
- Deve-se criar uma constante para definir o tamanho máximo permitido para a lista
- O primeiro elemento estará na primeira posição do vetor (posição zero)
- Deve-se criar uma variável para armazenar quantos elementos já foram inseridos

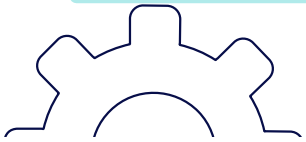


.cpp

```
Lista* criarLista(){
    Lista *lista = new Lista;
    if(!lista){
        cout << "Erro ao alocar memória!\n";
        exit(1);
    }
    lista->qtd = 0;
    return lista;
}
```

Main

```
Aluno alunoN;
Lista *lista;
lista = criarLista();
```



.h

```
struct ALUNO{
    int matricula;
    char nome[30];
    float n1, n2, n3;
};

struct LISTA{
   int qtd;
   ALUNO aluno[MAX];
};

typedef struct ALUNO Aluno;
typedef struct LISTA Lista;

Lista* criarLista();
```

.cpp

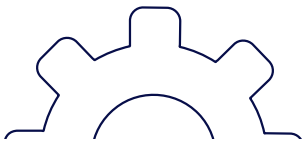
```
void liberarLista(Lista *lista){
    delete lista;
}
```

.h

```
void liberarLista(Lista *lista);
```

Main

```
    liberarLista(lista);
    return 0;
}
```



.h

```
int listaEhCheia(Lista *lista);
int listaEhVazia(Lista *lista);
```

.cpp

```
int listaEhCheia(Lista *lista){
    return (lista->qtd == MAX);
    //true (1) se estiver cheia
}

int listaEhVazia(Lista *lista){
    return (lista->qtd == 0);
    //true (1) se estiver vazia
}
```

Main

Essas funções são usadas nas funções de inserção e remoção.

.h

```
int tamanhoLista(Lista *lista);
```

.cpp

```
int tamanhoLista(Lista *lista){
    return lista->qtd;
}
```

Main

```
if(tamanhoLista(lista)==0)
    cout << "Não há aluno cadastrado!\n";
else
    cout << tamanhoLista(lista) << " aluno(s) cadastrado(s)!\n";
```

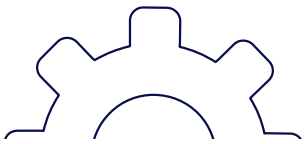
Inserção:
No final
No início
Ordenado

Inserção de Aluno no final da lista

```
int inserirFinal (Lista *lista, Aluno *alunoN){
    if(listaEhCheia(lista))
        return 0;
    lista->aluno[lista->qtd] = *alunoN;
    lista->qtd++;
    return 1;
}
```

Inserção de Aluno no início da lista

```
int inserirInicio (Lista *lista, Aluno *alunoN){
    if(listaEhCheia(lista))
        return 0;

    for(int i=lista->qtd;i>=0;i--)
        lista->aluno[i+1] = lista->aluno[i];
    lista->aluno[0] = *alunoN;
    lista->qtd++;
    return 1;
}
```



Inserção de Aluno Ordenado

```
int inserirOrdenado (Lista *lista, Aluno *alunoN){
    int p,i;
    if(listaEhCheia(lista))
        return 0;

    if(listaEhVazia(lista))
        lista->aluno[lista->qtd] = *alunoN;
    else{
        p =0;
        while(p<lista->qtd &&
              lista->aluno[p].matricula <= alunoN->matricula)
            p++;

        for(i=lista->qtd-1;i>=p;i--)
            lista->aluno[i+1] = lista->aluno[i];
        lista->aluno[p] = *alunoN;
    }
    lista->qtd++;
    return 1;
}
```

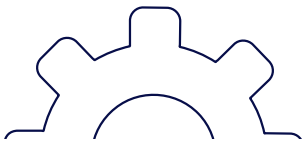
Remoção:
No final
No início
Ordenado

Remoção de Aluno no final da lista

```
int removerFinal(Lista *lista){
    if(!lista)
        return 0;
    if(lista->qtd==0)
        return 0;
    lista->qtd --;
    return 1;
}
```

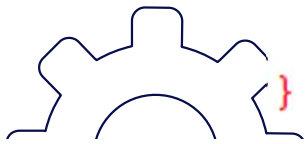
Remoção de Aluno no início da lista

```
int removerInicio(Lista *lista){
    if(!lista)
        return 0;
    if(lista->qtd==0)
        return 0;
    for(int i=0;i<lista->qtd -1;i++)
        lista->aluno[i] = lista->aluno[i+1];
    lista->qtd --;
    return 1;
}
```



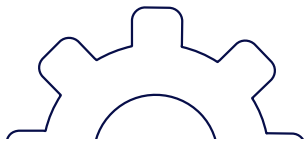
Remoção de um Item

```
int removerItem (Lista *lista, int matA){
     int p,i;
     if(!lista)
         return 0;
    if(lista->qtd==0)
         return 0;

    for(p=0; p<lista->qtd; ){
         if(lista->aluno[p].matricula != matA)
             p++;
    }
    if(p==lista->qtd)
         return 0; // não tem o elemento na lista

    for(i=p;i<lista->qtd-1;i++)
         lista->aluno[i] = lista->aluno[i+1];
    lista->qtd--;
    return 1;
}
```



Exibir dados da Lista

```
void exibirDadosLista(Lista *lista){
    int i;
    if(lista->qtd==0)
        cout << "Não existe aluno cadastrado na lista!\n";
    else{
        for (i=0;i<lista->qtd;i++){
            cout<< "\nNome: " << lista->aluno[i].nome;
            cout<< "\nMatrícula: " <<lista->aluno[i].matricula;
            cout<< "\nNotas: " << lista->aluno[i].n1 << " "
                << lista->aluno[i].n2 << " " << lista->aluno[i].n3 << endl;
        }
    }
}
```



Referências

EDELWEISS, N,; GALANTE, R.. Estruturas de dados. Porto Alegre: Bookman, 2009.

SZWARCFITER, J. L.; MARKENZON, L. Estruturas de dados e seus algoritmos. 3. ed. Rio de Janeiro: LTC, 2010.

ZIVIANI, N. Projeto de algoritmos: com implementações em Pascal e C. 3. ed. rev. e ampl. São Paulo: Cengage Learning, 2011.

Aulas e vídeo aulas do professor André Backes: https://www.facom.ufu.br/~backes/