**Estudo Dirigido**

**Listas – Implementação com arranjo**

Resolver em sala, individualmente ou em grupo de até três alunos, e apresentar os resultados ao professor, até o dia **28/03/2019**.

**Objetivo do Trabalho:** desenvolver habilidade de abstração e utilizar a abordagem de gerencia de memória com arranjo para implementar uma TAD Lista.

# **Introdução**

Nas aulas anteriores foram vistos os conceitos de Lista Linear e um *TADLista* para abstrair e implementar uma lista. Além da estrutura da lista, foram desenvolvidas as operações de inclusão no final, busca e exclusão pelo nome. O professor apresentou uma solução na Linguagem C para o problema do cadastro de alunos, apresentado no preâmbulo da apostila *Listas Lineares – Implementação com Arranjo*.

No presente estudo dirigido serão implementadas mais algumas operações do *TADLista*, já parcialmente implementado anteriormente, e modificar a aplicação *ListaAlunos* para incluir funconalidades para testar as novas operações do *TADLista*. Para recordar, o *TADLista* possui a seguinte estrutura de dados:

typedef struct {

Aluno alunos[MAX];

int tamanho;

}Lista;

E até aqui as seguintes operações:

void criaLista(Lista \*lst)– inicia uma lista vazia.

int vaziaLista(Lista \*lst)- verifica se uma lista está vazia.

int cheiaLista(Lista \*lst)- verifica se uma lista está cheia.

void insereLista(Aluno a, Lista \*lst, int \*codErro) – insere o aluno ***a*** no final da lista.

int buscaLista(Aluno a, Lista \*lst) – pesquisa se aluno ***a*** está na lista.

void excluiLista(Aluno a, Lista \*lst, int \*codErro) – remove o aluno ***a*** da lista.

void imprimeLista(Lista \*lst) – mostra todos os elementos da lista.

Neste estudo dirigido deverão implementadas as seguintes operações adicionais para o TAD Lista:

void insereNaPos (int i, Aluno a, Lista \*lst, int \*codErro) – insere o aluno ***a*** na posição ***i***, se válida.

int excluiDaPos(int i, Lista \*lst, int \*codErro) – remove da lista o elemento na posição ***i***, se válida.

void modifica(int i, Aluno a, Lista \*lst, int \*codErro) – atualiza os dados do aluno na posição ***i***, se válida.

Lista ordenaLista(Lista lst) – Ordena os elementos da Lista de acordo com um campo genérico. Retorna nova lista, preservando a lista original, que é passada por valor.

# **Roteiro do que deve ser feito**

O grupo deve seguir as instruções adiante para elaboração do trabalho:

**1 – Detalhamento e implementação do método:**

**void insereNaPos(int i, Aluno a, Lista \*lst, int \*codErro)**

Esta operação consiste em receber uma posição ***i*** da lista e inserir o Aluno ***a*** nesta posição, se válida. O algoritmo de inserção é o seguinte:

1. Se lista cheia, encerra e retorna código de erro pertinente;
2. Se posição não válida, encerra e retorna código de erro pertinente;
3. Sendo a posição válida e lista não cheia:
   1. Move todos os elementos desde a última posição até a posição i uma casa à direita no vetor;
   2. Insere o Aluno a na posição ***i*** do vetor;
   3. Incrementa o tamanho da lista.

**2 – Detalhamento e implementação do método:**

**int excluiDaPos(int i, Lista \*lst, int \*codErro)**

Esta operação consiste em receber uma posição ***i*** da lista excluir o Aluno que se encontra nesta posição, se válida. O algoritmo de exclusão é o seguinte:

1. Se lista vazia, encerra e retorna código de erro pertinente;
2. Se posição não válida, encerra e retorna código de erro pertinente;
3. Sendo a posição válida e lista não vazia:
   1. Move todos os elementos desde a posição posterior a ***i*** até a o último elemento uma casa à esquerda no vetor;
   2. Decrementa o tamanho da lista.

**3 – Detalhamento e implementação do método:**

void modifica(int i, Aluno a, Lista \*lst, int \*codErro)

Esta operação consiste em atualizar os dados do aluno na posição ***i*** da lista, se válida, com os dados recebidos pelo parâmentro Aluno ***a***. O algoritmo de atualização é o seguinte:

1. Se lista vazia, encerra e retorna código de erro pertinente;
2. Se posição não válida, encerra e retorna código de erro pertinente.
3. Sendo a posição válida e lista não vazia:
   1. Sobrepõe o Aluno a na posição i da lista;

Observe que esta operação modifica o elemento i da lista mas não altera o tamanho desta!

**4 – Detalhamento e implementação do método:**

Lista ordenaLista(Lista lst)

Esta operação consiste em gerar uma nova lista ordenada por um ou mais atributos. No caso, deverá ordenar pelo NOME do Aluno. A operação gera uma nova lista apenas para fins de apresentação, de modo a preservar a lista original sem modificação. Como sugestão, ordenar a lista pelo método da bolha, conforme algoritmo a seguir:

**Função** ordenaLista(Lista lst): Lista

**Var** aluno: Aluno

i, j: inteiro

**Inicio**

**Para** i **de** 0 **até** lst.tamanho – 2 **faça**

**Para** j **de** i+1 **até** ; lst.tamanho – 1 **faça**

**se**(**strcmp(lst.alunos[j].nome, lst.alunos[i].nome) < 0**) **Entao**

aluno = lst.alunos[i];

lst.alunos[i] = lst.alunos[j];

lst.alunos[j] = aluno;

**FimSe**

**FimPara**

**FimPara**

Retorne (lst);

**FimFunção**

Após a implementação das operações detalhadas acima, modificar a aplicação ***ListaAlunos.c*** de modo a criar novas funcionalidades para testar as operações recém criadas no *TADLista*. Deverão ser criadas as seguintes opções:

**Opção 4 – Inserir em posição informada**: semelhante a opção 1, porém deve obter também a posição que o usuário deseja inserir o aluno. Chamar a operação **insereNaPos** para efetuar inserção.

**Opção 5 – Excluir de posição informada**: semelhante a opção 2, porém deve solicitar a posição para exclusão e não o nome do aluno a excluir. Chamar a operação **excluiDaPos** para efetuar a exclusão.

**Opção 6 – Modificar nota**: esta opção faz o seguinte:

1. Obtém nome do aluno a modificar nota;
2. Chama a operação **buscaLista** para encontrar o aluno informado;
3. Se não encontrou o aluno, apresenta *msg* pertinente para usuário;
4. Se encontrou o aluno:
   1. Mostra a nota do aluno encontrado;
   2. Solicita a nova nota;
   3. Chama a operação **modifica** para atualizar a nota.

**Opção 7 – Listar Alunos ordenado por nome:** esta opção cria uma nova variável do tipo Lista e chama a operação **ordenaLista** para atualizar esta. Depois chama a operação imprimeLista para mostrar os alunos ordenados pelo nome.

# **Conclusão**

O trabalho deverá ser implementado em **C** e apresentado ao professor para avaliação, no prazo definido. Dúvidas, especialmente as relacionadas com *sintaxe* da programação em C, poderão ser dirimidas com o professor. Pontos omissos na especificação e decisões de implementação deverão ser resolvidos pelos alunos e devidamente justificados na apresentação do projeto.