

ESTRUTURA DE DADOS

A ESTRUTURA DE DADOS – LISTA

Olá!

Ao final desta aula, você será capaz de:

1. Compreender o conceito da Estruturas de Dados – Lista Linear;
2. Identificar as diferenças entre Lista Linear Sequencial e Encadeada;
3. Compreender e usar a Estruturas de Dados – Lista Linear Sequencial;
4. Identificar as principais características da Lista Linear Sequencial;
5. Compreender e usar várias operações realizadas com Lista Linear Sequencial;
6. Aplicar os conceitos de ordenação e pesquisa com Lista Linear Sequencial.

1 A lista abre as portas para as outras Estruturas de Dados

Na primeira aula, vimos que a família do Sr. Augusto quando fez o check-in no aeroporto, a atendente verificou se os nomes dele e dos parentes estavam na lista de passageiros.

A palavra Lista faz parte do nosso cotidiano, pois quantas vezes não fizemos uma Lista de compras antes de irmos ao supermercado ou uma Lista de convidados para uma festa de aniversário ou uma Lista de afazeres antes de viajarmos? Podemos concluir que uma Lista é, sem dúvida, a forma mais simples de agruparmos dados.

Na aula de hoje, faremos um estudo sobre as Listas Lineares e aprenderemos a construir vários trechos que serão úteis para as outras estruturas que também iremos estudar.



A estrutura que permite representar um conjunto de dados de forma a preservar a relação de ordem linear (ou total) entre eles é a lista linear. Uma lista linear é composta de nós, os quais podem conter, cada um deles, um dado primitivo ou um dado composto. (VELOSO,P.,SANTOS,C., AZEREDO,P., FURTADO, A., 1983,79)

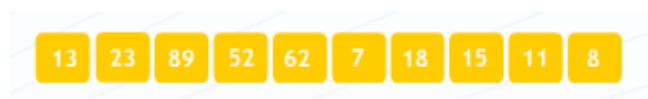
Todas as Estruturas de Dados têm suas importâncias e aplicações, mas, normalmente, começamos pelas Listas porque uma das formas mais simples de representá-las é com estruturas já conhecidas como as matrizes homogêneas e as matrizes heterogêneas. Além disso, elas servirão de base para que possamos implementar Estruturas de Dados que iremos estudar nas próximas aulas tais como Pilhas e Filas.

Da definição dos autores, vamos destacar alguns conceitos:

1) Nó ou nodo – é um item da lista.



2) Comprimento ou tamanho de uma lista – números de nós de uma LISTA. Exemplo de uma LISTA de comprimento 10 porque tem 10 nós.

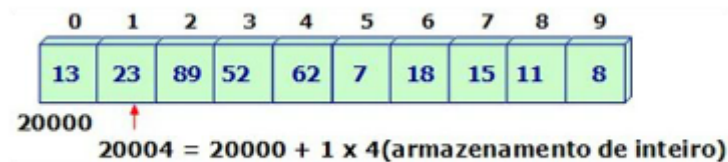


3) Se “uma Lista é composta de nós” então, uma Lista Vazia é uma lista que não tem nós.

Os elementos de uma Lista Linear podem ser agrupados de duas formas:

Sequencial

Esse tipo de estrutura apresenta os nós em posições contíguas de memória, isto é, um após o outro como já vimos quando estudamos matrizes na disciplina de Algoritmos. Sendo assim, fica fácil identificarmos o endereço de qualquer nó de uma Lista Sequencial.



Encadeado

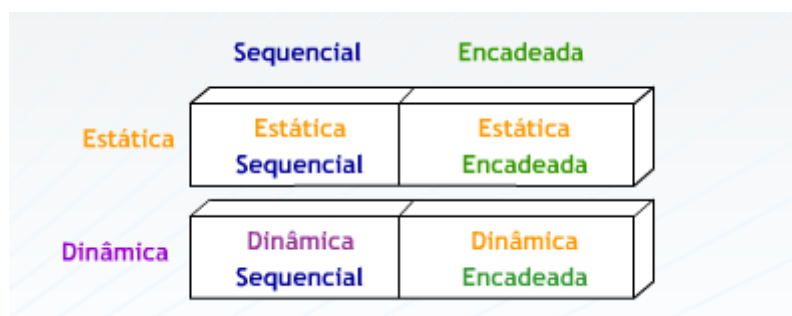
Esse tipo de estrutura será visto em uma aula, mas você já teve um contato com ela na primeira aula quando Sr, Augusto fez aquela brincadeira com os netos onde foram colocados bilhetes e, em cada bilhete, além de algo que eles tinham para retirar, existia uma referência para um outro local onde estava o próximo bilhete.

Este é princípio básico da **Lista Encadeada**, isto é, um conjunto de nós (nodos) encadeados onde cada nó contém o dado e um apontamento para o próximo nó, visto que eles não estão alocados de forma contígua.

Esta classificação que acabamos de estudar diz respeito à forma como os dados são armazenados na Memória Principal. Entretanto, não pode ser confundido com Alocação Estática ou Alocação Dinâmica.

Alocação Estática é determinada durante a compilação e é o que estamos fazendo e iremos fazer até à aula 7. Já na Alocação Dinâmica, você poderá determinar o tamanho, ou acrescentar, durante a execução.

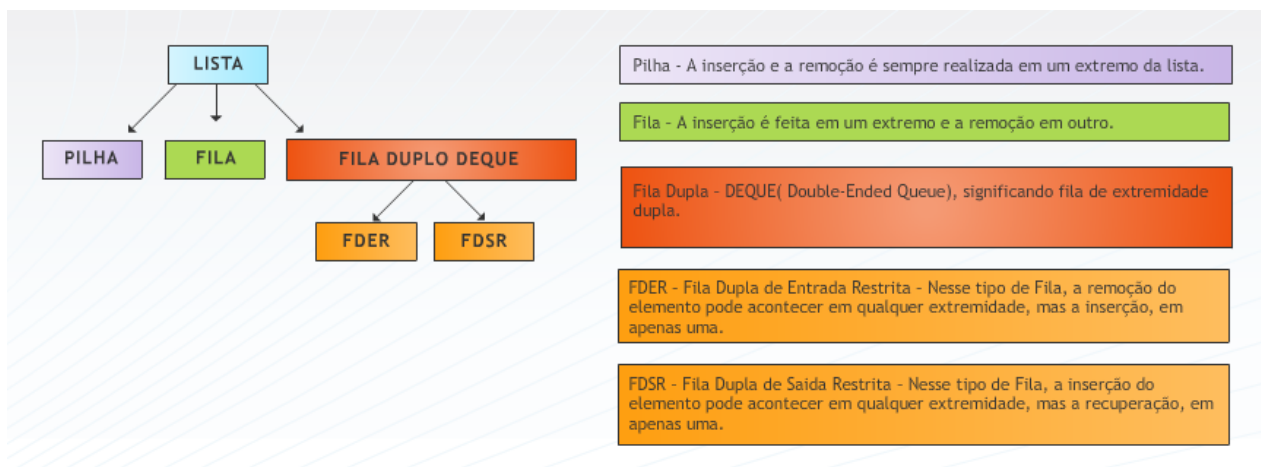
Reunindo estas classificações, podemos concluir que existem quatro possibilidades de Alocação de Memória.



Sobre este assunto, gostaria de colocar meu ponto de vista, pois não estou aqui para dizer para você o que é comum, mas o que é possível porque nunca saberemos o que um cliente pode desejar ou o que você poderá um dia precisar.

Na aula de hoje, não falaremos sobre Alocação Dinâmica. Sendo assim, nos restringiremos à *Lista Linear Sequencial*.

A Lista Linear Sequencial é ideal para um conjunto pequeno de dados. Apresenta a vantagem do acesso ao nó pelo índice e a desvantagem da movimentação quando se remove dado e insere se desejarmos deixar os vazios ao final. A forma de acesso à LISTA no que diz respeito à inserção e à remoção de um dado determina uma classificação mostrada na figura abaixo.



Nós podemos realizar várias operações com as *Listas*, mas, antes de qualquer operação, precisaremos preparar a *Lista* para receber os dados. Quando falamos em preparar, estamos nos referindo ao processo de inicialização da *Lista* que não poderá ser confundido com o fato de zerarmos todos os elementos de um vetor que servirá como um vetor acumulador. Em um dos exemplos, faremos isso, mas embora tenha o mesmo nome, o processo não é o mesmo.

Inicializar uma Lista é atribuir zero à variável que será responsável por armazenar a quantidade de dados que já se encontram na Lista.

Exemplos de algumas operações que podem ser realizadas com uma *Lista Linear Sequencial*.

- Criar uma Lista;
- Verificar se a Lista está vazia;
- Verificar se a Lista está cheia;
- Inserir elemento na Lista;
- Remover elemento da Lista;
- Exibir o tamanho da lista;
- Exibir a Lista;

- Exibir frequência;
- Pesquisar um elemento na Lista;
- Retornar a posição de um elemento da Lista;
- Alterar um elemento da Lista;
- Ordenar a Lista;
- Inserir ordenado na Lista;
- Concatenar Lista;
- Dividir Lista.

Evidentemente, é quase improvável que todos esses trechos estejam presentes em uma aplicação ao mesmo tempo e, por essa razão, resolvi apresentar quatro aplicações, procurando incluir a maioria dos trechos citados.

Gostaria que você tivesse clareza de que existe uma grande probabilidade de quando estivermos estudando esta aula que eu venha a achar que as soluções apresentadas poderiam estar muito melhores, mas isso é normal porque quando construímos um algoritmo somos influenciados pelo momento que estamos vivendo e, embora estejamos investidos em fazer sempre o melhor, às vezes, não acontece. Ainda bem que existe a manutenção para resolver esses e outros problemas.

Vamos aos exemplos!!!

EXEMPLO 1

Este exercício terá uma Lista com 5 nós para você poder testar. Os elementos desta LISTA serão inteiros e códigos de produtos. Além da opção de reiniciar a Lista para que você possa fazer vários testes, foram colocadas, no menu, 4 opções: Inserir elementos na Lista, Exibir os elementos da Lista, Exibir um elemento da Lista e Exibir o tamanho da Lista. Para os três primeiros, foram criadas funções, mas, para o último, por ser extremamente simples, não.

Inserir Código na LISTA

```
insere(codigoProduto,tam, 5);
```

```
void insere(int codigo[], int &t, int tamanho)
{
    int prod;
    if (tamanho == t)
        cout << "\nAtencao! Lista cheia\n";
    else
    {
        cout << "\nDigite codigo do produto a ser inserido: ";
        cin >> prod;
        codigo[t] = prod;
        t++;
    }
}
```

Esta função recebe o endereço do vetor, por referência o endereço da variável *t* e o *tamanho* do vetor. Inicialmente, é feito o teste para verificar se a Lista está cheia, visto que a variável *t* contém o tamanho atual e a variável *tamanho* contém o tamanho atual da *Lista*.

Caso a Lista não esteja cheia, é solicitada a digitação do código do produto e, como o tamanho atual corresponde também à posição que deverá ser armazenado o dado, o código digitado é colocado na Lista e o tamanho da *Lista* é incrementado. Como a variável *t* foi passada por referência, o valor se altera na função *main()*.

Exibir Lista

```
exibe(codigoProduto,tam);
```

```
void exibe(int codigo[], int t)
{
    int x;
    if (t == 0)
        cout << "\nAtencao! Lista vazia\n";
    else
        cout<< "\nRelacao dos Codigos\n";
        for(x = 0; x < t; x++)
            cout << "\n" << codigo[x];
}
```

Esta é uma função que recebe o endereço do vetor código e o tamanho atual da Lista. O teste verifica se a Lista está vazia e, caso não esteja, exibe os elementos que já se encontram armazenados.

Exibe um elemento da LISTA

```
elemento(codigoProduto, tam);
```

```
void elemento(int codigo[],int t)
{
    int prod,posicao;
    if(t == 0)
        cout << "\nAtencao! Lista vazia\n";
    else{
        cout << "\nQual a posicao que deseja? ";
        cin >> posicao;
        posicao--;
        if (posicao >= t)
            cout << "\nAtencao! Nenhum codigo armazenado ou posicao inexistente\n";
        else
            cout << "\nCodigo : " << codigo[posicao]<<"\n";
    }
}
```

Esta é uma função que recebe o endereço do vetor código e o tamanho atual da Lista. O teste verifica se a Lista está vazia e, caso não esteja, solicita ao usuário a posição que ele deseja do código a ser exibido, testando para garantir que o usuário digitou uma posição válida.

```
//Inicialização  
tam=0; // tam é o tamanho da matriz
```

Prepara a Lista para receber dados.
Trecho obrigatório antes de todos.

Aplicação


```

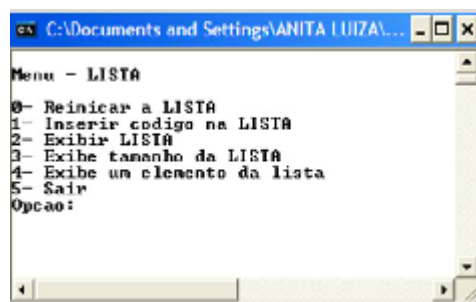
1 #include <iostream>
2 using namespace std;
3 void insere(int codigo[], int &t, int tamanho);
4 void exibe(int codigo[], int t);
5 void elemento(int codigo[],int t);
6 int main()
7 {
8     int tam, codigoProduto[5],op;
9
10    //Inicialização
11    tam = 0;
12    do
13    {
14        system("cls");
15        cout<<"\nMenu - LISTA \n";
16        cout<<"\nC- Reiniciar a LISTA";
17        cout<<"\n1- Inserir código na LISTA";
18        cout<<"\n2- Exibir LISTA";
19        cout<<"\n3- Exibe tamanho da LISTA";
20        cout<<"\n4- Exibe um elemento da lista";
21        cout<<"\n5- Sair";
22        cout<<"\nOpcao: ";
23        cin>>op;
24        system("cls");
25        switch(op)
26        {
27            case 0: //reinicialização
28                tam = 0;
29                break;
30
31            case 1: insere(codigoProduto,tam, 5);
32                break;
33
34            case 2: exibe(codigoProduto,tam);
35                break;
36
37            case 3: cout<<"\nTamanho da Lista: "<<tam;
38                break;
39
40            case 4: elemento(codigoProduto, tam);
41                break;
42
43            case 5: cout<<"\nFinalizando o programa da LISTA\n";
44                break;
45
46            default: cout<<"\nOpcao invalida\n";
47        }
48        cout<<"\n\n"; system("pause");
49    }while(op !=5);
50 }
51
52 void insere(int codigo[], int &t, int tamanho)
53 {
54     int prod;
55     if (tamanho == 0)
56         cout << "\nAtencao! Lista cheia\n";
57     else
58     {
59         cout << "\nDigite código do produto a ser inserido: ";
60         cin >> prod;
61         codigo[t] = prod;
62         t++;

```

```

63     }
64 }
65
66 void exibe(int codigo[], int t)
67 {
68     int x;
69     if (t == 0)
70         cout << "\nAtencao! Lista vazia\n";
71     else
72         cout<<"\nRelacao dos Codigos\n";
73     for(x = 0; x < t; x++)
74         cout << "\n" << codigo[x];
75 }
76
77 void elemento(int codigo[],int t)
78 {
79     int prod,posicao;
80     if(t == 0)
81         cout << "\nAtencao! Lista vazia\n";
82     else
83     {
84         cout << "\nQual a posicao que deseja? ";
85         cin >> posicao;
86         posicao--;
87         if (posicao >= t)
88             cout<<"\nAtencao! Nenhum codigo armazenado cu posicao";
89         else
90             cout << "\nCodigo: " << codigo[posicao]<<"\n";
91     }
92 }

```



EXEMPLO 2

Este exercício terá uma *Lista* com 5 nós para você poder testar. Os elementos desta Lista serão inteiros e matrículas (suponha de funcionários de uma empresa de pequeno porte). Além da opção de reiniciar a Lista para que você possa fazer vários testes, foram colocadas, no menu, 4 opções: Inserir matricula na Lista, Exibir Lista, Procurar matricula na Lista e Remover matricula da Lista. Para todos, foram criadas funções.

Inserir matrícula na LISTA

```
cout << "\nDigite matricula a ser inserida: ";
cin >> mat;
insere(matricula,mat,tam, 5);
```

```
void insere(int matric[], int m, int &t, int tamanho)
{
    if (tamanho == t)
        cout << "\nAtencao! Lista cheia\n";
    else
    {
        matric[t] = m;
        t++;
    }
}
```

Esta função parece diferir da primeira apenas no fato da pergunta sobre a matrícula a ser inserida, não estar dentro da função. Entretanto, no processo de inserção, em um único momento, existe uma diferença que não é prejudicial. Você sabe qual é?

```
exibe(matricula,tam);
```

```
void exibe(int matric[], int t)
{
    int x;
    if (t == 0)
        cout << "\nAtencao! Lista vazia\n";
    else
        cout << "\nRelacao das Matriculas\n";
    for(x = 0; x < t; x++)
        cout << "\n" << matric[x];
}
```

Esta função é idêntica à do primeiro exemplo.

Procurar matrícula na LISTA

Somente a linha `posicao = buscaSequencial(matricula,mat,tam);` seria a chamada da função, mas se eu não colocasse todo o trecho, ficaria difícil entender o que se faz com o retorno da função.

Para procurar a matrícula, foi usada uma função que realiza uma busca sequencial que recebe o endereço de um vetor e o tamanho atual, ficando com três possibilidades de retorno: -1, se a Lista estiver vazia; -2 se a matrícula

não foi achada ou a posição onde se encontra a matrícula. A chamada da função é feita com retorno para uma variável que terá seu valor testado para que o usuário possa obter respostas diferenciadas de acordo com o retorno. O fato de encontrarmos a expressão `posicao + 1` é porque os índices, como sabemos, estão defasados de 1.

```
cout << "\nQual matrícula a ser procurada? ";
cin >> mat;
posicao = buscaSequencial(matricula,mat,tam);
if(posicao == -1)
    cout << "\nAtencao! Lista vazia\n";
else if (posicao == -2)
    cout << "\nAtencao! Matrícula não encontrada\n";
else
    cout << "\nMatrícula encontrada na posicao: " << posicao+1<<"\n";
```


```
int buscaSequencial(int matric[], int m, int t)
{
    int x;
    if (t == 0)
        return -1; // testa lista vazia
    for (x = 0; x < t; x++)
        if( matric[x] == m)
            return x; // retorna o deslocamento do endereço base
    return -2; // percorreu toda a lista e não achou
}
```

Remover matrícula da LISTA

```
cout << "\nQual matrícula a ser removida? ";
cin >> mat;
remove(matricula,mat,tam);
```

```
void remove(int matric[], int m, int &t)
{
    int pos,x;
    pos = buscaSequencial( matric, m, t);
    if (pos == -1) // testando se a lista esta vazia
        cout << "\nAtencao! Lista vazia\n";
    else if (pos == -2)
        cout << "\nAtencao! Matrícula não encontrada\n";
    else
    {
        // desloca todos os elementos da lista, se necessário, deixando vazios ao final
        for(x=pos; x< t-1; x++)
            matric[x] = matric[x+1];
        t--; // atualiza o tamanho da lista
        cout << "\nMatrícula Removida\n";
    }
}
```

Somente a linha **remove(matricula,mat, tam);** sena a chamada da função, mas coloquei as outras duas para você entender sem ter que ler o código todo. Para que possa remover uma matricula da *Lista*, a função **remove** chama a função **buscaSequencial**. Além de fazer os testes, ela desloca todas as matrículas para cima, mas isso não é obrigatório. Foi uma opção, pois você pode inserir o valor em qualquer posição.



```
//Inicialização  
tam=0; // tam é o tamanho da matriz
```

Prepara a Lista para receber dados.
Trecho obrigatório antes de todos.

Aplicação


```

1 #include <iostream>
2 using namespace std;
3 void insere(int matric[], int m, int &t, int tamanho);
4 void exibe(int matric[], int t);
5 int buscaSequencial(int matric[], int m, int t);
6 void remove(int matric[], int m, int &t);
7 int main()
8 {
9     int tam, matricula[5], mat, posicao, op;
10
11     //Inicialização
12     tam = 0;
13
14     do
15     {
16         system("cls");
17         cout<<"\nMenu 1 - LISTA \n";
18         cout<<"\n0- Reiniciar a LISTA";
19         cout<<"\n1- Inserir matricula na LISTA";
20         cout<<"\n2- Exibir LISTA";
21         cout<<"\n3- Procurar matricula na LISTA";
22         cout<<"\n4- Remover matricula da LISTA";
23         cout<<"\n5- Sair";
24         cout<<"\nOpcao: ";
25         cin>>op;
26         system("cls");
27         switch(op)
28         {
29             case 0: //reinicialização
30                 tam = 0;
31                 break;
32
33             case 1:
34                 cout << "\nDigite matricula a ser inserida: ";
35                 cin >> mat;
36                 insere(matricula, mat, tam, 5);
37                 break;
38
39             case 2: exibe(matricula, tam);
40                 break;
41
42             case 3:
43                 cout << "\nQual matricula a ser procurada? ";
44                 cin >> mat;
45                 posicao = buscaSequencial(matricula, mat, tam);
46                 if(posicao == -1)
47                     cout << "\nAtencao! Lista vazia\n";
48                 else if (posicao == -2)
49                     cout << "\nAtencao! Matricula nao encontrada\n";
50                 else
51                     cout << "\nMatricula encontrada na posicao: " << posicao+1<<"\n";
52                 break;
53
54             case 4:
55                 cout << "\nQual matricula a ser removida? ";
56                 cin >> mat;
57                 remove(matricula, mat, tam);
58                 break;
59
60             case 5: cout<<"\nFinalizando o programa da LISTA\n";
61                 break;
62
63         }
64     }

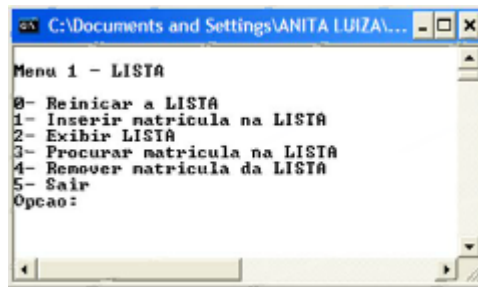
```



```

61
62     default: cout<<"\nOpcao invalida\n";
63 }
64 cout<<"\n\n"; system("pause");
65 }while(op !=5);
66 }
67
68 void insere(int matric[], int m, int &t, int tamanho)
69 {
70     cout << "\nAtencao! Lista cheia\n";
71 }
72 else
73 {
74     matric[t] = m;
75     t++;
76 }
77 }
78
79 void exibe(int matric[], int l)
80 {
81     int x;
82     if (t == 0)
83         cout << "\nAtencao! Lista vazia\n";
84     else
85         {cout<<"\nRelacao das Matriculas\n";
86           for(x = 0; x < l; x++)
87
88
89
90
91 int buscaSequencial(int matric[], int m, int t)
92 {
93     int x;
94     if (t == 0)
95         return -1; // testa lista vazia
96     for (x = 0; x < l; x++)
97         if( matric[x] == m)
98             return x; // retorna o deslocamento do endereco base
99     return -2; // percorreu toda a lista e nao achou
100 }
101
102 void remove(int matric[], int m, int &t)
103 {
104     int pos,x;
105     pos = buscaSequencial( matric, m, t);
106     if (pos == -1) // testando se a lista esta vazia
107         cout << "\nAtencao! Lista vazia\n";
108     else if (pos == -2)
109         cout << "\nAtencao! Matricula nao encontrada\n";
110     else
111     {
112         // desloca todos os elementos da lista, se necessario, deixando vazios ao final
113         for(x=pos; x< t-1; x++)
114             matric[x] = matric[x+1];
115         t--; // atualiza o tamanho da lista
116         cout << "\nMatricula Removida\n";
117     }
118 }

```

EXEMPLO 3

Este exercício terá uma Lista com 10 nós para você poder testar. Os elementos desta Lista serão inteiros e idades. Além da opção de reiniciar a Lista, foram colocadas, no menu, 4 opções: Inserir idade na LISTA, Exibir Lista, Ordenar LISTA e Exibir frequencia. Para todos, foram criadas funções.

Inserir idade na LISTA

```
cout << "\nDigite idade(10-19) a ser inserida na LISTA: ";
cin >> id;
while(id < 10 || id > 19)
{
    cout << "\nAtencao para o intervalo.";
    cout << "\nDigite idade(10-19) a ser inserida na Lista: ";
    cin >> id;
}
insere(idades, id, tam, 10);
f3=0;
```

```
void insere(int IDS[], int idade, int &t, int tamanho)
{
    if (tamanho == t)
        cout << "\nAtencao! Lista cheia\n";
    else
    {
        IDS[t] = idade;
        t++;
    }
}
```

A função `insere` é idêntica a dos outros exemplos, mas gostaria de fazer algumas considerações sobre o trecho que antecede à chamada da função, visto que coloquei uma proteção para garantir que somente as idades no intervalo de 10 - 19 serão aceitas. Suponha que este programa esteja realizando uma pesquisa que seleciona um grupo de atletas nesta faixa etária. Observe também que está presente uma variável de nome `f3` que recebe 0 toda vez que um número é inserido. Ela será testada em outro trecho. Fique atento!

Exibir Lista

```
exibe(idades,tam);
```

```
void exhibe(int IDS[], int t)
{
    int x;
    if (t == 0)
        cout << "\nAtencao! Lista vazia\n";
    else
        cout << "\nRelacao das Idades\n";
    for(x = 0; x < t; x++)
        cout << "\n" << IDS[x];
}
```

Esta função é idêntica à do primeiro exemplo.

Ordenar LISTA

Gostaria que ficasse atento ao trecho que chama a função seleção porque após terminar a ordenação, a variável f3 recebe o valor 1, significando que a Lista está ordenada que é uma condição necessária para a função que exhibe a ocorrência das idades no conjunto (frequência). Quanto ao método de ordenação, foi apenas uma escolha entre os três que estudamos.

```
//ordena Metodo por selecao. Poderia ser por insercao ou bolha
if (tam == 0)
    cout << "\nAtencao! Lista vazia\n";
else
{
    selecao(idades, tam);
    f3= 1;
}
```

```
void selecao(int IDS[], int t)
{int i, j, menor, temp;
 if (t == 0)
    cout << "\nAtencao! Lista vazia\n";
 else
 { for(i=0; i<t-1; i++)
 {
    menor=i;
    for(j=i+1; j<t; j++)
        if(IDS[j] < IDS[menor])
            menor=j;
    temp=IDS[i];
    IDS[i]=IDS[menor];
    IDS[menor]=temp;
 }
 }
 cout<< "\nLista Ordenada\n";
}
```

Exibir frequência

```
if(f3==1)
    frequencias(idades,tam);
else
    cout<<"\nOrdene primeiro a LISTA\n";
```

```
//precisa estar ordenado
void frequencias(int IDS[], int t)
{
    int i, c, frequencia[]={0,0,0,0,0,0,0,0,0,0};
    if (t== 0)
        cout << "\nAtencao! Lista vazia\n";
    else
    {
        for(i=0;i<t;i++)
        { frequencia[IDS[i]-10]++;
          cout<<"\n"<<IDS[i]<<"\t"<< frequencia[IDS[i]-10]; //SO PARA FINS DIDATICOS
        }
        // exibe frequencia dos números sem repetição
        cout<<"\n\nIdade\tFrequencia\n";
        cout<<"\n"<<IDS[0]<<"\t"<< frequencia[IDS[0]-10];
        for(i=1; i<t; i++)
            if (IDS[i] != IDS[i-1] )
                cout<<"\n"<<IDS[i]<<"\t"<< frequencia[IDS[i]-10];
    }
}
```

Começo analisando a chamada da função frequências, pois ela está dentro de um teste feito com a variável f3, uma vez que ela sinaliza se a Lista está, ou não ordenada. Deixando mais claro: toda vez que uma idade é inserida, f3 recebe 0 e quando a Lista é ordenada, f3 recebe 1. Sendo assim, a função frequências só será executada se a Lista já estiver ordenada.

A função é bem simples e creio que a linha que vou analisar é que poderá trazer alguma dúvida.

frequencia[IDS[i]-10]++;

Se somente idades no intervalo de 10 a 19 serão armazenadas no vetor IDS, podemos associar o vetor frequência da seguinte maneira:

Posição 0 com a idade 10

Posição 1 com a idade 11

...

Posição 8 com a idade 18

Posição 9 com a idade 19

Como? Se você subtrair 10 da idade, obterá a posição e poderá incrementar direto o vetor frequência.

Gostou? Mas isto foi possível porque o vetor era inteiro e uma expressão foi conseguida.

```
//Inicialização  
tam=0; // tam é o tamanho da matriz
```

Prepara a Lista para receber dados.
Trecho obrigatório antes de todos.

Aplicação

```
#include <iostream>

using namespace std;

void insere(int IDS[], int idade, int &t, int tamanho);

void exhibe(int IDS[], int t);

void selecao(int IDS[], int t);

void frequencias(int IDS[], int t);

int main()

{

int tam, op, f3=0, idades[10], id;

//Inicialização

tam = 0;

do

{

system("cls");

cout<<"\nMenu 2 - LISTA \n";

cout<<"\n0- Reiniciar a LISTA";

cout<<"\n1- Inserir idade na lista";

cout<<"\n2- Exibir lista";

cout<<"\n3- Ordenar lista";

cout<<"\n4- Exibe frequencia";

cout<<"\n5- Sair";

cout<<"\nOpcao: ";

cin>>op;

system("cls");

switch(op)
```

```

{
case 0: //reinicialiação
tam = 0;
break;
case 1:
cout << "\nDigite idade(10-19) a ser inserida na Lista: ";
cin >> id;
while(id<10 || id >19)
{
cout<<"\nAtencao para o intervalo.";
cout<<"Digite idade(10-19) a ser inserida na Lista: ";
cin>>id;
}
insere(idades,id,tam, 10);
break;
case 2: exhibe(idades,tam);
break;
case 3: //ordena Metodo selecao poderia ser insercao ou bolha
selecao(idades, tam);
f3=1;
break;
case 4:if(f3==1)
frequencias(idades,tam);
else
cout<<"\nOrdene primeiro o vetor\n";
break;
case 5: cout<<"\nFinalizando o programa da LISTA\n";
break;
default: cout<<"\nOpcao invalida\n";
}
cout<<"\n\n"; system("pause");
}while(op !=5);

```

```

}

void insere(int IDS[], int idade, int &t, int tamanho)

{
if (tamanho == t)

cout << "\nAtencao! Lista cheia\n";

else

{

IDS[t] = idade;

t++;

}

}

void exibe(int IDS[], int t)

{

int x;

if (t == 0)

cout << "\nAtencao! Lista vazia\n";

else

for(x = 0; x < t; x++)

cout << "\n" << IDS[x];

}

void selecao(int IDS[], int t)

{int i, j, menor, temp;

if (t == 0)

cout << "\nAtencao! Lista vazia\n";

else

{ for(i=0; i<t -1; i++)

{

menor=i;

for(j=i+1; j<t ; j++)

if(IDS[j] < IDS[menor])

menor=j;

temp=IDS[i];

```

```

IDS[i]=IDS[menor];
IDS[menor]=temp;
}
}

cout<<"\nLista Ordenada\n";

}

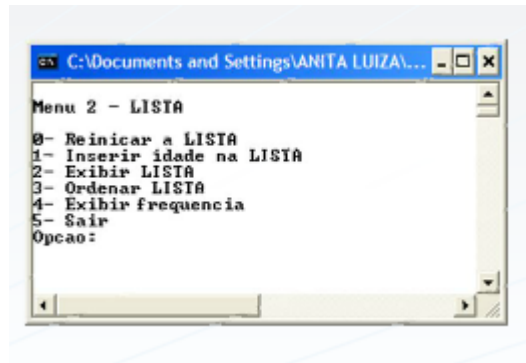
void frequencias(int IDS[], int t)
{
int i, c, frequencia[]={0,0,0,0,0,0,0,0,0,0};
if (t == 0)
cout << "\nAtencao! Lista vazia\n";
else
{
for(i=0;i<t;i++)
{ frequencia[IDS[i]-10]++;
cout<<"\n"<<IDS[i]<<"\t"<< frequencia[IDS[i]-10]; //SÓ PARA FINS DIDÁTICOS
}

// exibe frequência dos números sem repetição
cout<<"\n\nIdade\tFrequencia\n";

cout<<"\n"<<IDS[0]<<"\t"<< frequencia[IDS[0]-10];

for(i=1; i<t; i++)
if (IDS[i] != IDS[i-1] )
cout<<"\n"<<IDS[i]<<"\t"<< frequencia[IDS[i]-10];
}
}
}

```



EXEMPLO 4

Este exercício terá uma Lista com 5 nós para você poder testar. Os elementos desta Lista serão structs que contém nome e CR. Além da opção de reiniciar a Lista, foram colocadas, no menu, 3 opções: Inserir dados Ordenados pelo nome na LISTA, Exibir Lista e Alterar CR. Para todos, foram criadas funções. Além dessas, foram criadas duas outras funções.

Inserir dados ordenados pelo nome na LISTA

Esta função possibilita que os dados fiquem armazenados ordenados pelo nome. O teste da *estrutura while* faz isto quando usa a função `strcmp(...)` comparando o membro nome com o nome procurado pelo usuário. Observe o deslocamento para baixo conseguido com a estrutura do `for` e a vantagem de se usar uma matriz de struct quando precisamos mover várias variáveis.

Os três testes são necessários porque precisamos verificar se a Lista está cheia ou se é o primeiro a inserir, visto que terá um tratamento especial ou será qualquer outro elemento da *Lista*. Os dados que serão inseridos na Lista por esta função foram lidos por duas outras funções que chamei de funções auxiliares porque a função `alteraCR` tem bem irá precisar delas.

insereOrdenado(alunos,n, nt, tam, 5);

```
void insereOrdenado(dado AL[],char nome[], float CR, int &t, int tamanho)
{
    int i, c;
    if (tamanho == t)
        cout << "\nAtencao! Lista cheia\n";
    else if (t > 0)
    {
        // Procura a posição de inserção
        c = 0;
        while (strcmp(nome, AL[c].nome) > 0 && c < t)
            c++;
        // Desloca os structs para baixo
        for (i = t; i >= c + 1; i--)
            AL[i] = AL[i-1];
        // Copia no struct
        AL[c].CR = CR;
        strcpy(AL[c].nome, nome);
        t++; // Incrementa a quantidade de nós na lista
    }
    else
    {
        AL[0].CR = CR;
        strcpy(AL[0].nome, nome);
        t++;
    }
}
```

Exibir Lista

exibe(alunos,tam);

```
void exibe(dado AL[], int t)
{
    int x;
    if (t == 0)
        cout << "\nAtencao! Lista vazia\n";
    else
        cout << "\nCR\tNome\n";
    for (x = 0; x < t; x++)
        cout << "\n" << AL[x].CR << "\t" << AL[x].nome;
}
```

Esta função é idêntica à do primeiro exemplo

Alterar CR

A função alteraCR poderia ter sido implementada de várias maneiras, mas achei que deveria simplificá-la, transformando em funções dois trechos que existiam dentro dela e que eram os mesmos da função insereOrdenado. Sendo assim, as duas se tornaram mais legíveis. Como no outro exemplo tinha usado a pesquisa sequencial, usei a pesquisa binária e, desta vez, com nome. Já falamos sobre a comparação de vetores de char e a necessidade do uso da função strcmp(...) para comparar vetores de char e da função strcpy(...) para copiar valores em vetores de char. Se ainda tiver dúvida, releia as aulas anteriores.

alteraCR(alunos,tam);

```
void alteraCR(dado AL[],int t)
{ int fim, meio, inicio=0;
  char procura[31];
  float nt;
  if (t == 0)
    cout << "\nAtencao! Lista vazia\n";
  else
  {
    entraNome(procura);
    fim = t - 1; //Inicio da pesquisa binária
    meio = (inicio + fim) / 2;
    while(strcmp(procura,AL[meio].nome) != 0 && inicio != fim)
    {
      if(strcmp(procura,AL[meio].nome) > 0)
        inicio = meio + 1;
      else
        fim = meio;
      meio = (inicio + fim) / 2;
    }
    if(strcmp(AL[meio].nome,procura) == 0)
    {
      cout << "\nCr atual: " << AL[meio].CR << endl;
      nt = entraCR();
      AL[meio].CR = nt;
    }
    else
      cout << "\nNome não encontrado\n";
  }
}
```

entraNome();

Acredito que você tenha muitas perguntas sobre esta função e, por isso, vou me antecipar.

- 1) Foi criada a variável lixo, muito maior do que o necessário, porque o usuário não "atende" aos nossos pedidos.
- 2) cin.get(); está presente porque esta função estará em um case do switch, implicando que houve uma leitura e o buffer tem armazenado o código da tecla enter. Já falamos que esta é uma das formas de "limpar" um caracter.
- 3) O trecho de proteção feito com while assegura nomes com até 30 caracteres.
- 4) Após o trecho de proteção, é copiado o nome digitado com a garantia de não ter mais de 30 caracteres para a variável nome.
- 5) A estrutura do for, tendo no corpo um comando de atribuição que agrega função toupper, possibilitará a conversão para maiúscula de todos os caracteres.

entraNome(n);

```
void entraNome(char nome[])
{
    int x;
    char lixo[100];
    cin.get();
    cout << "\nNome ate 30 caracteres: ";
    cin.getline(lixo, 100);
    while(strlen(lixo)>30)
    {
        cout<< "\nExcede numero de caracteres permitido.";
        cout<< "\nNome ate 30 caracteres: ";
        cin.getline(lixo, 100);
    }
    strcpy(nome,lixo);
    for(x=0; x<strlen(lixo); x++)
        nome[x]=toupper(nome[x]);
}
```

nt=entraCR();

A função entraCR() é uma função que retorna o CR e por essa razão, usei o comando de atribuição para armazenar o retorna em uma variável. FO trecho de proteção garante que somente CRs ente 0 a 10 inclusive serão armazenados.

```
float entraCR()
{
    float nt;
    cout << "\nCR: ";
    cin>>nt;
    while(nt<0 || nt>10)
    {
        cout<< "\nCR invalido";
        cout<< "\nCR: ";
        cin>>nt;
    }
    return nt;
}
```

//Inicialização
tam=0; // tam é o tamanho da matriz

Prepara a Lista para receber dados.
Trecho obrigatório antes de todos.

Aplicação


```

#include <iostream>
#include <cstring>
using namespace std;
struct dado
{
    char nome[31];
    float CR;
};
void entraNome(char nome[]);
float entraCR();
void insereOrdenado(dado AL[],char nome[], float CR, int &t, int tamanho);
void exhibe(dado AL[], int t);
void alteraCR(dado AL[], int t);
void remove(dado AL[], int &t);

int main()
{
    dado alunos[5];
    int tam, op, f3=0 ;
    float nt;
    char n[31];
    //Inicialização
    tam=0;
    do
    {
        system("cls");
        cout<<"\nMenu 3 - LISTA \n";
        cout<<"\n0- Reiniciar a LISTA";
        cout<<"\n1- Inserir dados Ordenados pelo nome na LISTA";
        cout<<"\n2- Exibir LISTA";
        cout<<"\n3- Alterar CR";
        cout<<"\n4- Sair";
        cout<<"\nOpcao: ";
        cin>>op;
        system("cls");
        switch(op)
        {
            case 0: //reinicialização
                tam = 0;
                break;
            case 1:
                entraNome(n);
                nt=entraCR();
                insereOrdenado(alunos,n, nt, tam, 5);
                break;

            case 2: exhibe(alunos,tam);
                break;

            case 3:
                cout << "\nNome ate 30 caracteres: ";
                cin.getline(lixo, 100);
                while(strlen(lixo)>30)
                {
                    cout<<"\nExcede numero de caracteres permitido.";
                    cout<<"\nNome ate 30 caracteres: ";
                    cin.getline(lixo, 100);
                }
                strcpy(nome,lixo);
                for(x=0; x<strlen(lixo); x++)
                    nome[x]=toupper(nome[x]);
                break;
            case 4:
                break;
        }
    }
}

```

```

float entraCR()
{
    float nt;
    cout << "\nCR: ";
    cin>>nt;
    while(nt<0 || nt>10)
    {
        cout<<"\nCR invalido";
        cout<<"\nCR: ";
        cin>>nt;
    }
    return nt;
}

void insereOrdenado(dado AL[],char nome[], float CR, int &t, int tamanho)
{
    int i, c;
    if (tamanho == t)
        cout << "\nAtencao! Lista cheia\n";
    else if(t>0)
    {
        // Procura a posição de inserção
        c = 0;
        while (strcmp(nome ,AL[c].nome)>0 && c < t)
            c++;
        // Desloca os structs para baixo
        for (i = t; i >= c+1; i--)
            AL[i] = AL[i-1];
        // Copia no struct
        AL[c].CR = CR;
        strcpy(AL[c].nome,nome);
        t++; // Incrementa a quantidade de nós na lista
    }
    else
    {
        AL[0].CR = CR;
        strcpy(AL[0].nome,nome);
    }
    t++;
}

void exibe(dado AL[], int t)
{
    int x;
    if (t == 0)
        cout << "\nAtencao! Lista vazia\n";
    else
        cout<<"\nCR\tNome\n";
    for(x = 0; x < t; x++)
        cout << "\n" << AL[x].CR<<"\t"<<AL[x].nome;
}

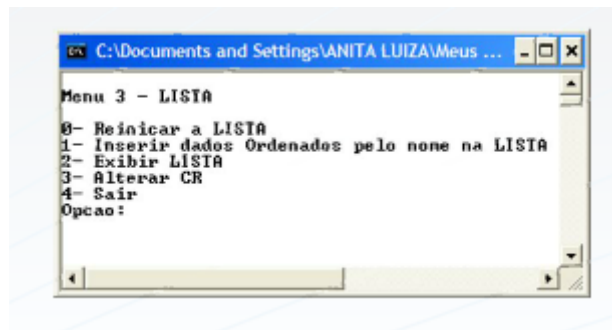
void alteraCR(dado AL[],int t)
{
    int fim, meio, inicio=0;
    char procura[31];
    float nt;
    entraNome(procura);
    fim= t - 1;
    meio=(inicio+fim)/2;
    while(strcmp(procura,AL[meio].nome)!=0 && inicio != fim)
    {
        if(strcmp(procura,AL[meio].nome)>0)
            inicio=meio+1;
    }
}

```

```

else
    fim=meio;
    meio=(inicio+fim)/2;
}
if(strcmp(AL[meio].nome,procura)==0)
{
    cout<<"\nCr atual: "<<AL[meio].CR<<endl;
    nt=entraCR();
    AL[meio].CR=nt;
}
else
    cout<<"\nNome nao encontrado\n";
}

```



Já falei que é um grande desafio apresentar os conceitos das Estruturas de Dados num curso à distância e, a cada aula, vai ficando mais difícil porque vou me colocando no seu lugar e me deparo com tantos conceitos que precisaria aprender se tivesse do outro lado que vou procurando ficar mais detalhista e criando mais desenhos na esperança de me comunicar com você de todas as formas possíveis.

Tentei com esses exemplos mostrar que as aulas anteriores vão se agregando às novas aulas. Além disso, gostaria que percebesse que o tamanho dos nossos códigos fontes já se tornou muito maiores.

Falei no início que a parceria Algoritmos e Estruturas de Dados é necessária para fazer de você um programador.

Então, tudo precisa ser muito bem assimilado e, para que isso se torne possível, a dedicação será fundamental.

Não deixe pendente nenhum assunto. Tire suas dúvidas com seu professor. Faça todos os exercícios e leia várias vezes as aulas, complementando com o material de apoio e pesquisando sobre o assunto na biblioteca virtual e na internet.

Até a próxima aula.

O que vem na próxima aula

Na próxima aula, você aprenderá os seguintes assuntos:

- A estrutura de Dados Pilha;
- Representar a estrutura de dados pilha por contiguidade;
- Implementar operações com pilhas.

CONCLUSÃO

Nesta aula, você:

- Compreendeu a importância da Estrutura de Dados Lista Linear Sequencial;
- Aprendeu a diferença entre Lista Linear Sequencial e Lista Linear Encadeada;
- Compreendeu e usou várias operações que podem ser feitas com a Lista Linear Sequencial;
- Aplicou os conceitos de ordenação e pesquisa com Lista Linear Sequencial.