

ESTRUTURA DE DADOS

LISTAS ENCADEADAS – FINALIZANDO / PILHAS E FILAS DINÂMICAS

Olá!

Ao final desta aula, você será capaz de:

1. Compreender e implementar operações com lista linear simplesmente encadeada, realizando aplicações;
2. Compreender e implementar operações com pilha dinâmica;
3. Compreender e implementar operações com fila dinâmica.

1 Encadear para Dinamizar

Corrigindo as Tarefas da aula anterior:

Na Aula 8, deixei uma tarefa para você. Gostaria de lembrar que a solução que irei apresentar não significa a única correta. A sua poderá estar melhor do que a minha. Sempre digo que não existe uma única solução. Mas, vamos lá.

1.1 Listando a lista

Trecho do programa original	Função
<pre>//listando c=1; cout<<"\n\nUsando uma estrutura de Repeticao\n"; while(lista) { cout<<"\nValor do "<<c<<"o no: "<<lista->num; lista=lista->prox; c++; }</pre>	<pre>void exibe(nodo*lista) { int c=1; cout<<"\n\nListando\n"; while(lista) { cout<<"\nValor do "<<c<<"o no: "<<lista->num; lista=lista->prox; c++; } }</pre>

Analisando...

- 1- Por que mudei o título? Achei que para a função, seria melhor Listando, mas isso é indiferente.
- 2- Por que foi declarada com int a variável c na função? No programa ela também estava declarada, mas lá no início.

1.2 Removendo elemento da lista

Trecho do programa original	Função
<pre>//removendo nó aux = lista; lista = lista->prox; delete aux;</pre>	<pre>nodo *remove(nodo *lista) { nodo *aux; aux = lista; lista = lista->prox; delete aux; return lista; }</pre>

Analisando...

- 1- Por que mudei o título? Achei que para a função, seria melhor Listando, mas isso é indiferente.
- 2- Por que foi declarada com int a variável c na função? No programa ela também estava declarada, mas lá no início.
- 3- Por que apareceu return na função e não tinha no trecho original? No programa original, só tínhamos a função main e, sendo assim, todas as variáveis se encontravam na mesma da memória.

1.3 Inserindo elemento da Lista

Trecho do programa original	Função
<pre>//inserir nó na Frente temp = new nodo; temp->num = 23; temp->prox = lista; lista = temp;</pre>	<pre>nodo* insereFrente(nodo &ptr, int valor) { nodo*temp = new nodo; temp->num = valor; temp->prox = ptr; return temp; }</pre>

Esta foi a função que criei na aula passada e, agora, ao clicar no link, podemos ver como ficou o código com as três funções:

http://estaciadocente.webaula.com.br/cursos/gon119/docs/ExercicioAula8_AULA9.pdf

Nesta aula, daremos continuidade ao estudo de Listas Encadeadas e, como não existe restrição sobre onde se insere e onde se remove em uma Lista, fiz um menu com uma grande miscelânea que acredito que ninguém irá me copiar na prática, pois meu objetivo foi puramente didático.

Gostaria de ter implementado mais funções, mas, não tínhamos tempo para absorver todas. Sendo assim, espero que vocês usem sua criatividade e depois as criem.

Por uma questão de coerência sobre tudo que foi dito nas Aulas 5, 6 e 7, aproveitei as funções usadas neste menu e implementei Pilha Dinâmica e a Fila Dinâmica.

Uma das vantagens de se implementar estruturas dinamicamente é o fato delas ocuparem espaço estritamente necessário.

Às vezes, um aluno me pede: me dá um exemplo bem simples que justifique usar uma estrutura dinamicamente. Então, falo sempre algo parecido com isso.

A cada ano, o número de alunos inscritos no ENEM cresce assustadoramente, uma vez que a oferta à Universidade Pública se tornou maior para os alunos que conseguirem uma boa pontuação nesse exame. Como determinar o número de fiscais, de salas e a quantidade de memória para armazenar os dados dos candidatos se não sabemos quantos são a priori? Esse é um caso típico do uso de uma estrutura de dados dinâmica, uma vez que ela pode aumentar ou diminuir de tamanho.

Vamos agora apresentar o menu deste programa e como o código é muito grande, você deverá clicar no link abaixo para visualizá-lo.



```
I:\AULA9\cpp\menuListaEncadeada.exe
< < < Alocacao Dinamica < < <
< 1- Insere Frente - Lista
< 2- Insere Fim - Lista
< 3- Remove Frente - Lista
< 4- Remove Fim - Lista
< 5- Exibe Lista
< 6- Substitui No na Lista
< 7- Conta Nos da Lista
< 8- Busca na Lista
< 9- Libera Lista
< 10- Sai
< Opcao:
< < < < < < < < < < < < <
```

Código fonte:

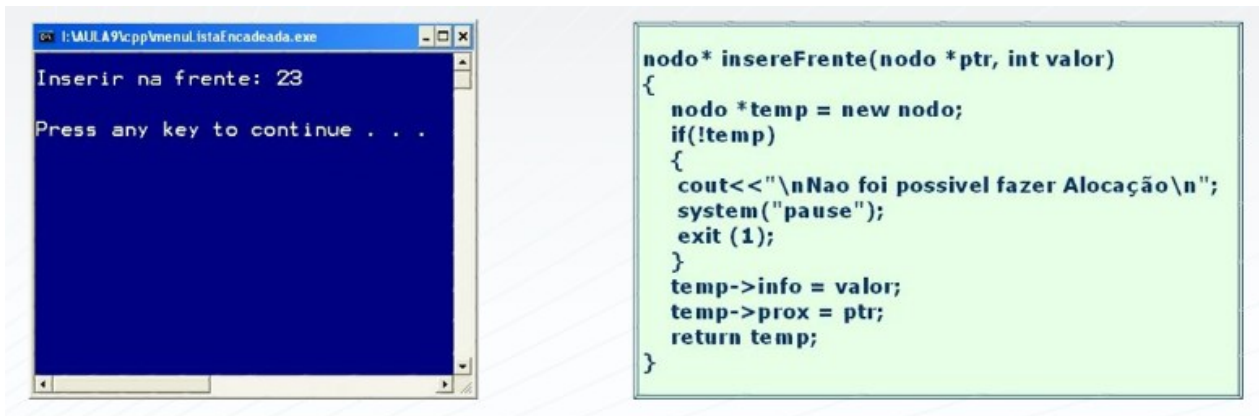
<http://estaciocente.webaula.com.br/cursos/gon119/docs/menuListaEncadeada.pdf>

2 As funções

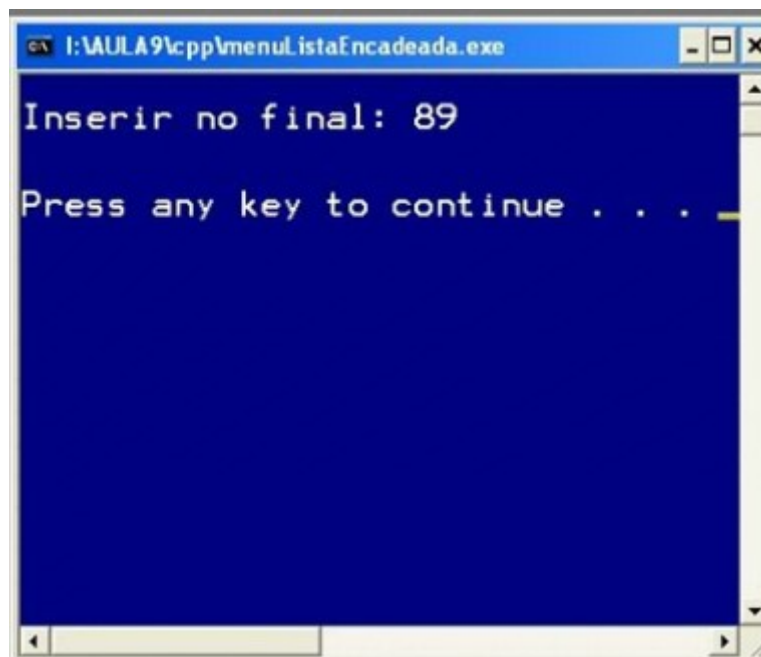
2.1 *insereFrente*

Analisando a função...

A função `insereFrente` já foi analisada na Aula 8.



2.2 *InsereFim*



```

nodo* insereFim(nodo *ptr, int valor)
{
    nodo *novo, *aux;
    novo = new nodo;
    if(!novo)
    {
        cout<<"\nNao foi possivel fazer Alocação\n";
        system("pause");
        exit (1);
    }
    novo->info = valor;
    novo->prox = NULL;
    if (!ptr)
        ptr = novo;
    else
    {
        aux = ptr;
        while (aux->prox)
            aux = aux->prox;
        aux->prox = novo;
    }
    return ptr;
}

```

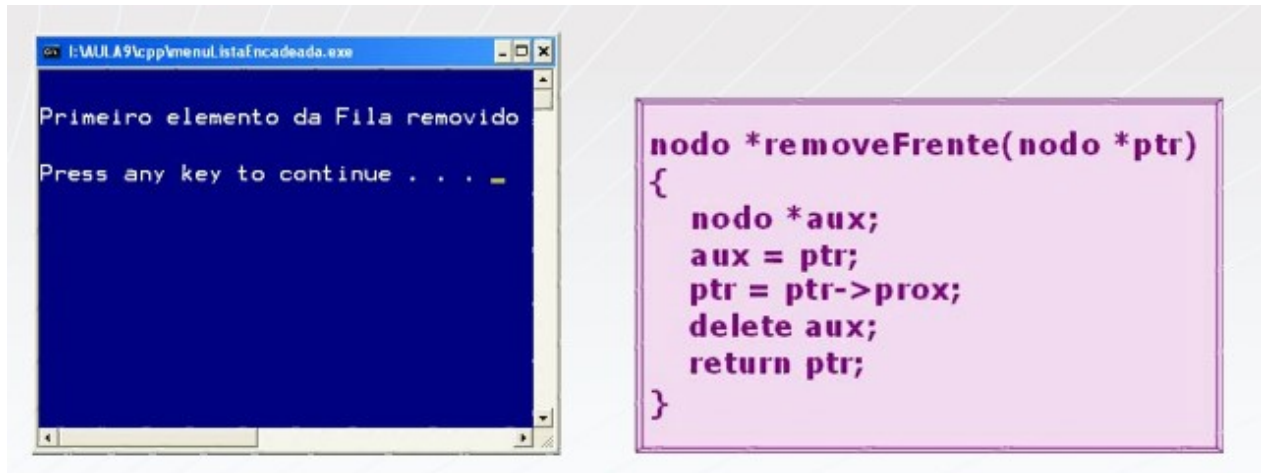
Analizando a função

- 1) São declaradas duas variáveis ponteiros do tipo nodo.
- 2) Um novo nó foi alocado dinamicamente através do comando: novo= new nodo;.
- 3) Logo depois, foi realizado um teste para saber se a alocação foi bem sucedida.
- 4) Tendo sido bem sucedida, trecho após o if, o membro info do nó de nome novo recebe o valor e o membro ponteiro, NULL.
- 5) O teste if(!ptr) significa a mesma coisa que if(ptr==NULL). Logo, se a Lista estiver vazia, o nó novo é inserido na Lista, mas se não for, toda a Lista é percorrida até chegar ao final, para poder inserir o novo nó.

Observe o while(aux->prox), significando enquanto tiver nó, com auxílio da variável nó aux, a Lista vai sendo percorrida, uma vez que aux=aux->prox; "assume" o nó atual e, no próximo teste do while, aux->prox, já estará apontando para o próximo nó.

Gostaria de enfatizar que os nós de uma lista não possuem nome como estamos acostumados com os elementos das matrizes. Para acessar um nó, precisamos fazer uso dos ponteiros e é por essa razão que para percorrer uma lista, foi criada esta variável aux que recebe o endereço de uma célula, permitindo que seu conteúdo seja acessado para listá-lo ou substituí-lo.

2.3 removeFrente



Analisando a função

- 1) O teste para verificar se a Lista estava vazia foi feito na função main que é uma opção para nem chamar uma outra função se não for necessário.
- 2) Você deve estar se lembrando do algoritmo que foi construído na Aula 8, se não estiver, dê uma olha lá, quando foi removido um nó da lista que se encontrava na frente, pois isso justifica a presença da variável aux para preservar o conteúdo da célula para que ela possa depois ser deletada.
- 3) Retornando um ponteiro para a Lista.

Observe a situação atual da lista após a remoção do primeiro elemento.



A screenshot of a Windows command prompt window. The title bar shows the file path: F:\ESTRUTURA_DE_DADOS\AULA9\cpp\bons\menuLi... The window has a blue background with white text. The text displayed is: "Listando", followed by the numbers "23", "89", and "52" on separate lines. Below these numbers is the prompt "Press any key to continue . . .".

```
F:\ESTRUTURA_DE_DADOS\AULA9\cpp\bons\menuLi...
Listando
23
89
52
Press any key to continue . . .
```

2.4 removeFim



A screenshot of a Windows command prompt window. The title bar shows the file path: I:\AULA9\cpp\menuListaEncadeada.exe. The window has a blue background with white text. The text displayed is: "Ultimo elemento da Fila removido", followed by the prompt "Press any key to continue . . .".

```
I:\AULA9\cpp\menuListaEncadeada.exe
Ultimo elemento da Fila removido
Press any key to continue . . .
```



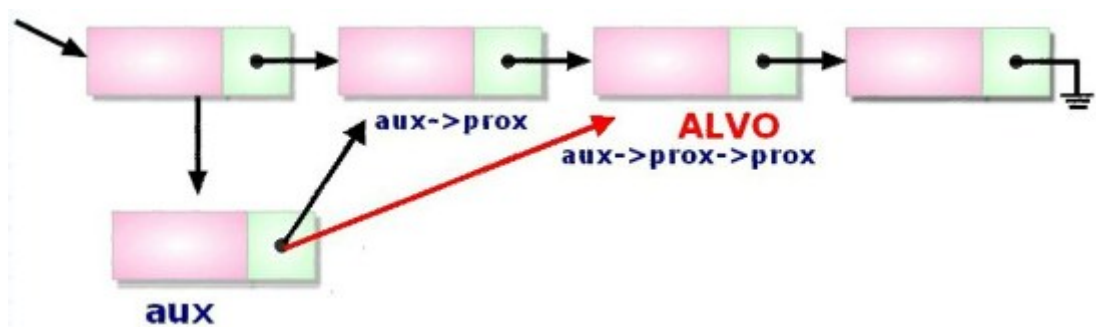
```

nodo* removeFim(nodo *ptr)
{
    nodo * aux, *ultimo;
    if (!ptr->prox)
    {
        delete ptr;
        ptr = NULL;
        return ptr;
    }
    else
    {
        aux = ptr;
        while (aux->prox->prox )
            aux = aux->prox;
        ultimo = aux->prox;
        delete ultimo;
        aux->prox = NULL;
        return ptr;
    }
}

```

Analisando a Função...

O princípio desta função é simples. Ela testa se só existe um nó, pois `!ptr->prox` significa "perguntar" se é nulo o apontamento, isto é, não existe outro nó e, caso exista, percorre a Lista, visando sempre o penúltimo. Por que o penúltimo? Onde você está vendo isso? Creio que essas perguntas estão vindo em sua cabeça agora. Vou tentar lhe explicar de uma forma bem simples, mas se você não entender, me pergunte novamente. O alvo é procurar o penúltimo para que possa remover o último. Por essa razão, a linha `aux->prox->prox` estará sempre olhando o próximo do próximo porque se ele não existir, significa que `aux->prox` é o último



2.5 *exibe*



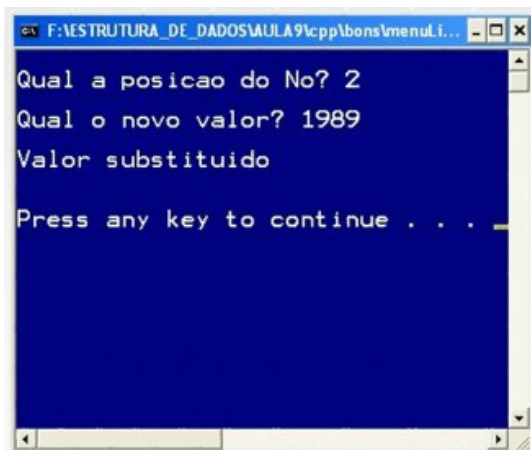
```
I:\AULA9\cpp\menuListaEncadeada.exe
Listando
23
89
Press any key to continue . . .
```

```
void exhibe(nodo *ptr)
{
    cout<<"\nListando\n";
    while(ptr)
    {
        cout<<"\n"<<ptr->info;
        ptr=ptr->prox;
    }
}
```

Analizando a função...

Esta é uma função simples porque enquanto existir Lista, while(ptr), exhibe o elemento e passa para o próximo nó.

2.6 *substituiNo*



```
F:\ESTRUTURA_DE_DADOS\AULA9\cpp\bons\menuLi...
Qual a posicao do No? 2
Qual o novo valor? 1989
Valor substituido
Press any key to continue . . .
```

```
void substituiNo(nodo *ptr, int posicao, int novoValor)
{
    int conta = 1;
    while (conta != posicao )
    {
        ptr = ptr->prox;
        conta++;
    }
    ptr->info = novoValor;
}
```

Analizando a função...

1) A função recebe o ponteiro para a Lista, a posição do nó a ser substituído e o valor. 2) O trecho é simples porque foi criada uma variável contadora para que possa ser comparada com a posição. Enquanto não chegar à posição pedida, a lista é percorrida.

3) Quando a posição for achada, a repetição é finalizada, e o valor atribuído.

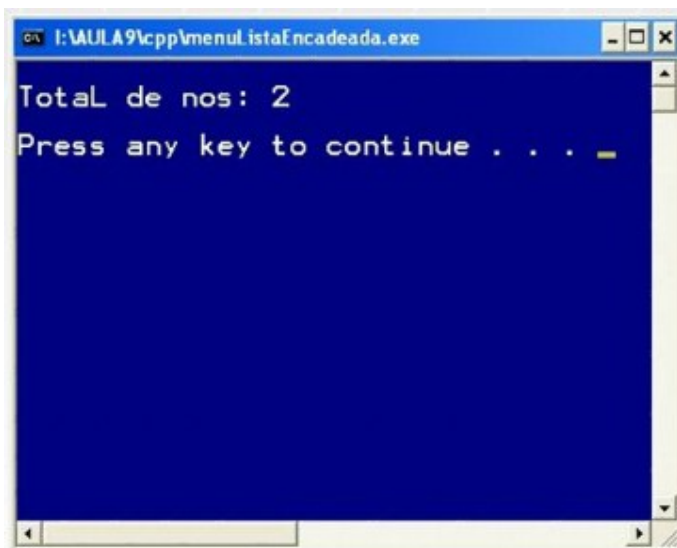
4) Não precisa ter retorno porque foi passado o endereço da Lista.

Foi feito um teste na função main para evitar que uma posição inexistente seja digitada, garantindo que a chamada da função só será feita quando existe o nó na Lista.

Veja como ficou a Lista depois da substituição.



2.7 ContaNos



```
int contaNos(nodo *ptr)
{
    int conta = 0;
    while (ptr)
    {
        conta++;
        ptr = ptr->prox;
    }
    return conta;
}
```

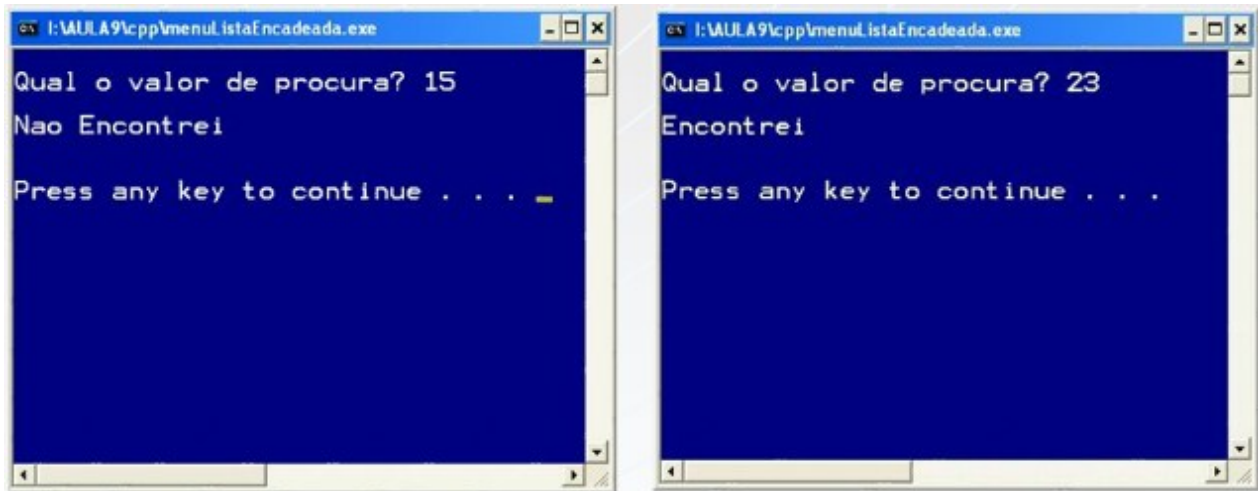
Analisando a função...

Você tem várias maneiras para contar nós de uma lista.

Poderá criar uma variável que será incrementada toda vez que um nó for criado e decrementada toda vez que for removido.

Resolvi apresentar desta forma para você treinar, mais uma vez, o percorrer de uma lista.

2.8 buscaSequencial

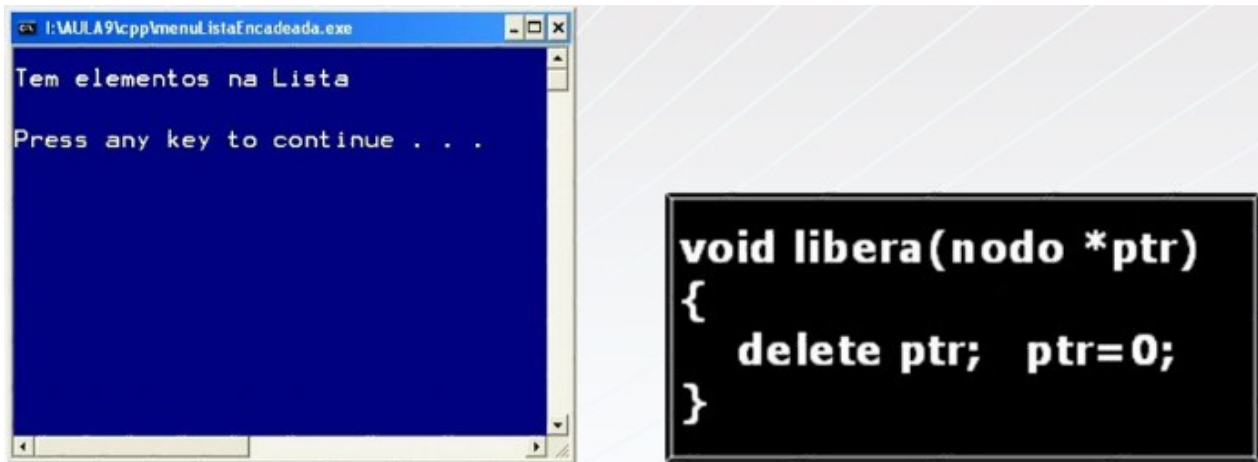


```
int buscaSequencial(nodo *ptr, int valor)
{
    while (ptr)
    {
        if (ptr->info == valor)
            return 1;
        ptr = ptr->prox;
    }
    return 0;
}
```

Analizando a função...

Mais uma vez a função percorre a Lista(ptr=ptr->prox) e repete esse processo enquanto existir Lista. Se encontrão valor, retoma 1, mas se não encontrar, retorna 0. Você pode usar esta função para contar quantos elementos existem na Lista iguais a um valor, por exemplo.

2.9 libera



Analizando a função...

Foi feita uma proteção para que não seja liberada enquanto tiver elementos na Lista porque sabemos que o melhor seria deletar todos os elementos antes. Entretanto, ficará muito cansativo remover um a um e por essa razão, sugiro que você pense em incluir um trecho que remova os que ainda existam na Lista antes de liberá-la.

3 Implementação da Pilha Dinâmica e da Fila Dinâmica

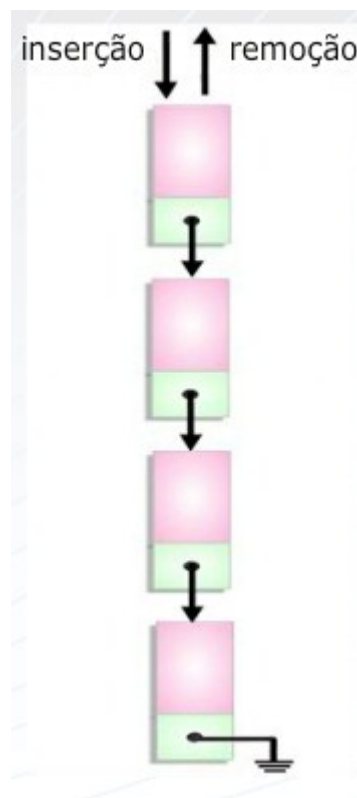
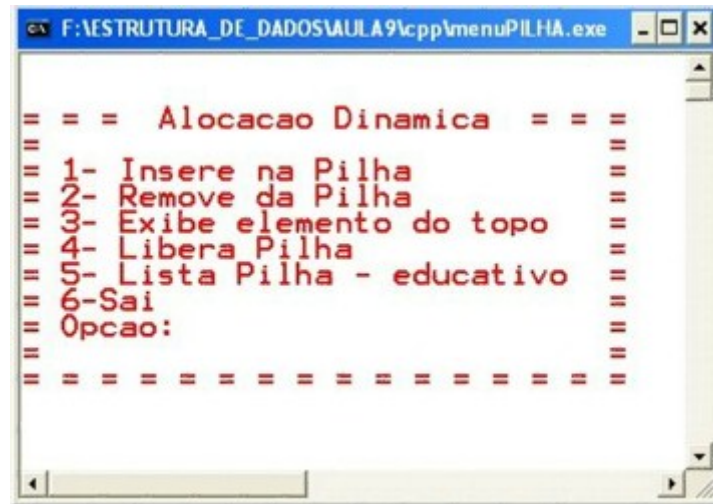
Depois de termos visto várias operações realizadas com as Listas Encadeadas e, como já estudamos nas Aulas 6 e 7 que tanto a Pilha quanto a Fila são casos particulares da Lista, resolvi não implementar nenhuma outra função embora pudesse trazer maior eficiência para os programas.

Meu objetivo, nesse caso, foi meramente didático, e talvez viesse a complicar o entendimento se introduzisse outras funções.

Tenho clareza de que a criação, principalmente para a Fila de um ponteiro indicador de fim e outro de início melhoraria significativamente o trecho de remover ao final, mas ficará para um outro momento. Afinal, essas aulas finais estão com conteúdos bem mais complicados do que as outras.

Como todas as funções que serão usadas na Pilha Dinâmica e na Fila Dinâmica já foram explicadas nesta aula, e como você já entende o conceito dessas estruturas, só me resta apresentar o programa e as funções que serão usadas de acordo com as características de cada estrutura.

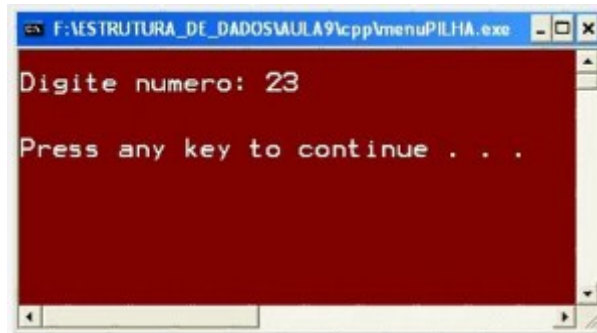
3.1 Pilha Dinâmica



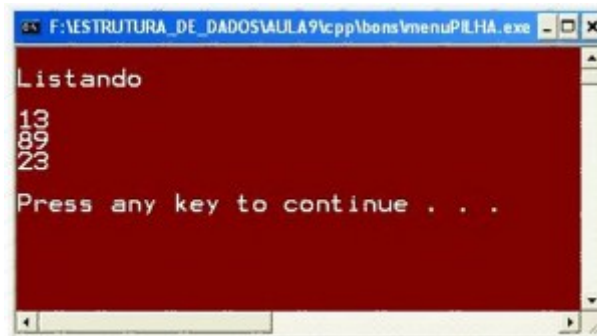
Vamos agora apresentar o menu deste programa. Clique no link a seguir para visualizá-lo: <http://estaciocente.webaula.com.br/cursos/gon119/docs/menuPilhaDinamica.pdf>

1) Insere na Pilha

InsereFrente



Suponha a inserção de mais dois elementos.



```
nodo* insereFrente(nodo *plista, int valor)
{
    nodo *temp = new nodo;
    if(!temp)
    {
        cout<<"\nNao foi possivel fazer Alocação\n";
        system("pause");
        exit (1);
    }
    temp->info = valor;
    temp->prox = plista;
    return temp;
}
```


2) Remove da Pilha

remove



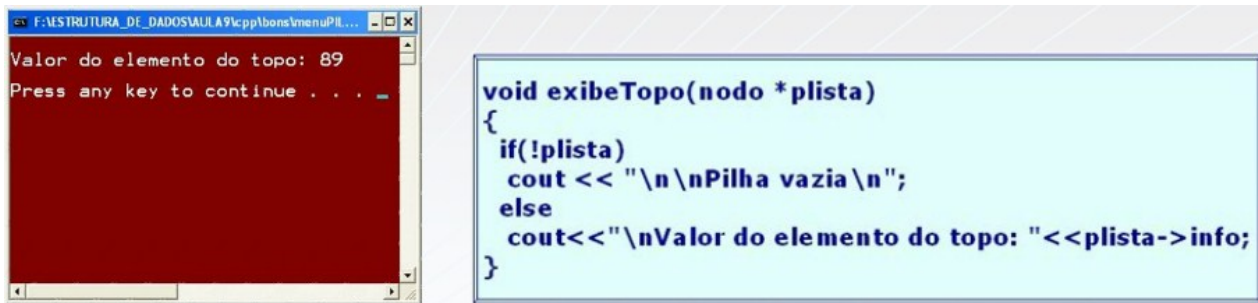
Após a remoção



```
nodo *remove(nodo *plista)  
{  
    nodo *aux;  
    aux = plista;  
    plista = plista->prox;  
    delete aux;  
    return plista;  
}
```

3) Exibe Elemento do Topo

exibeTopo

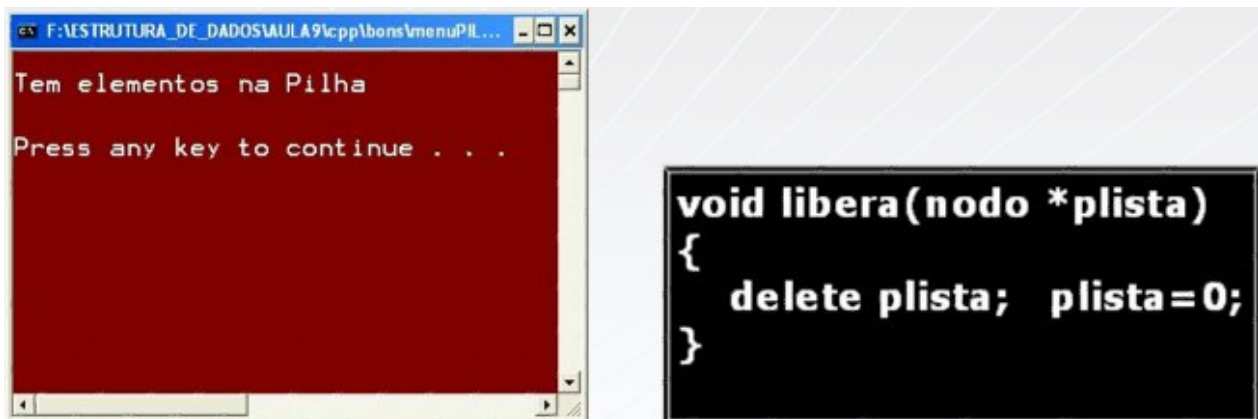


Analisando a função...

Como esta função não tínhamos visto na Lista, merece ser analisada. O ponteiro sempre aponta para o primeiro nó logo, é só pedir para exibir o membro sa struct que contem a informação.

4) Libera Pilha

libera



Analisando a função...

Foi feita uma proteção para que não seja liberada enquanto tiver elementos na Pilha porque sabemos que o melhor seria deletar todos os elementos antes. Entretanto, ficará muito cansativo remover um a um e por essa razão, sugiro que você pense em incluir um trecho que remova os que ainda existam na Pilha antes de liberá-la.

5) exibeLista

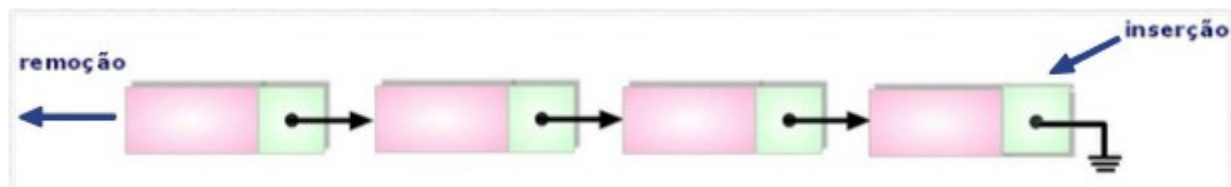
```
F:\ESTRUTURA_DE_DADOS\AULA9\cpp\bons\menuPILHA.exe
Listando
89
23
Press any key to continue . . . _
```

```
void exibeLista(nodo *plista)
{
    cout<<"\nListando\n";
    while(plista)
    {
        cout<<"\n"<<plista->info;
        plista=plista->prox;
    }
}
```

3.2 Fila Dinâmica

Vamos agora apresentar o menu deste programa e como o código é muito grande, você deverá clicar no botão ao lado para visualizá-lo.

```
F:\ESTRUTURA_DE_DADOS\AULA9\cpp\bons\menuFILA.exe
- - -   Alocação Dinâmica   - - -
- 1- Insere na Fila          -
- 2- Remove da Fila         -
- 3- Exibe primeiro da Fila -
- 4- Libera Fila             -
- 5- Lista Fila - educativo  -
- 6- Sai                     -
- Opcao:                     -
- - - - - - - - - - - - -
```

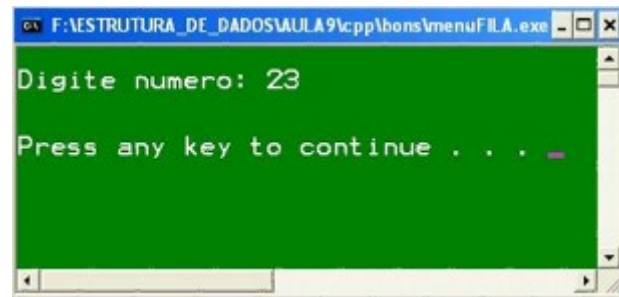


Código fonte:

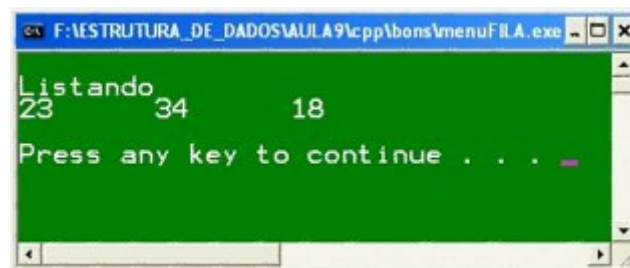
<http://estaciadocente.webaula.com.br/cursos/gon119/docs/menuFilaDinamica.pdf>

1) Insere na Fila

insereFim



Suponha que você tenha digitado mais dois números. Veja como ficou sua Lista.



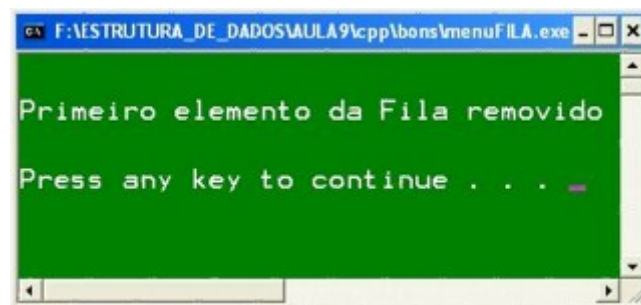
```

nodo *insereFim(nodo *plista, int valor)
{
    nodo *novo, *aux;
    novo = new nodo;
    if(!novo)
    {
        cout<<"\nNao foi possivel fazer Alocao\n";
        system("pause");
        exit (1);
    }
    novo->info = valor;
    novo->prox = NULL;
    if (plista == NULL)
        plista = novo;
    else
    {
        aux = plista;
        while (aux->prox )
            aux = aux->prox;
        aux->prox = novo;
    }
    return plista;
}

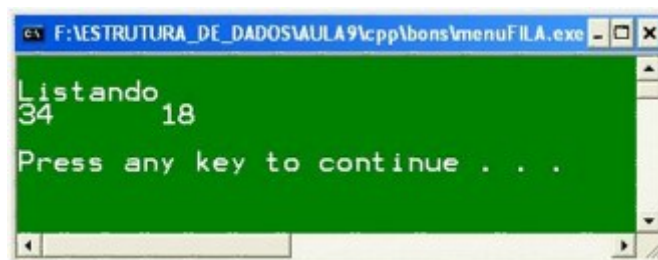
```

2) Remove na Fila

remove



Agora sua lista após a remoção



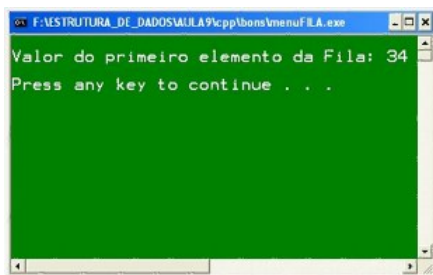
```

nodo *remove(nodo *plista)
{
    nodo *aux;
    aux = plista;
    plista = plista->prox;
    delete aux;
    return plista;
}

```

3) Exibe Primeiro da Fila

exibePrimeiro



```

void exibePrimeiro(nodo *plista)
{
    if(!plista)
        cout << "\n\nFila vazia\n";
    else
        cout<< "\nValor do primeiro elemento da Fila: "<<plista->info;
}

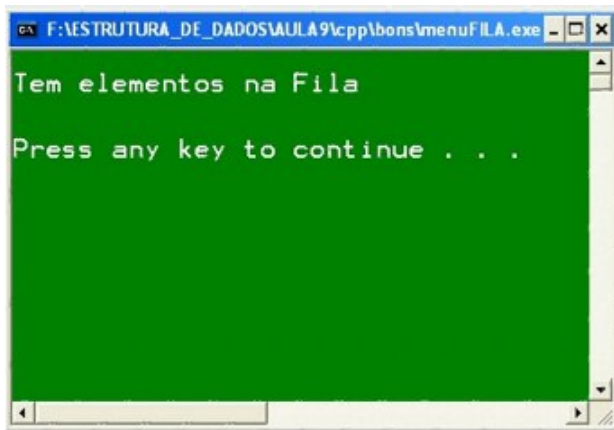
```

Analisando a função...

Essa função é idêntica à função `exibeTopo` da Pilha.

4) Libera Fila

libera



```
void libera(nodo *plista)
{
    delete plista;  plista=0;
}
```

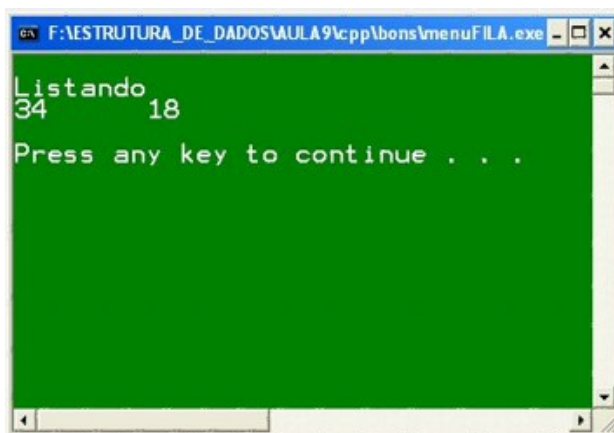
Analizando a função...

Foi feita uma proteção para que não seja liberada enquanto tiver elementos na Fila porque sabemos que o melhor seria deletar todos os elementos antes.

Entretanto, ficará muito cansativo remover um a um e por essa razão, sugiro que você pense em incluir um trecho que remova os que ainda existam na Fila antes de liberá-la.

5) Lista Fila

exibeLista



```
void exibeLista(nodo *plista)
{
    cout<<"\nListando\n";
    while(plista)
    {
        cout<<plista->info<<"\t";
        plista=plista->prox;
    }
}
```

Analizando a função...

Se você observar, só alterei o \n para \t para exibir em uma linha.

4 Finalizando

Na minha visão, as Aulas 8 e 9 trouxeram um conjunto de informações muito importante para vocês e não tenho a pretensão de que tudo será absorvido de uma única vez.

Espero que vocês releiam estas aulas várias vezes e, com certeza, a cada leitura, a compreensão se tornará maior. Procurei deixar esta aula mais de aplicação, visto que todos os conceitos já tinham sido ministrados nas aulas 5, 6, 7 e 8. Havia necessidade de ser uma aula prática porque sinto que vocês têm que visualizar todo o conjunto e só mostrar as funções em separado não oferece embasamento para implementação de códigos maiores. Se tiverem alguma dúvida, conversem com seu professor e até a próxima aula.

O que vem na próxima aula

Na próxima aula, aprenderemos os seguinte assunto:

- Listas Duplamente Encadeadas

CONCLUSÃO

Nesta aula, você:

- Completou o estudo sobre Lista Simplesmente Encadeada;
- Compreendeu e usou Pilha Dinâmica;
- Compreendeu e usou Fila Dinâmica.