

ESTRUTURA DE DADOS

ALOCAÇÃO DINÂMICA / LISTAS ENCADEADAS – INTRODUÇÃO

Olá!

Nesta aula, você irá:

1. Conceituar ponteiro;
2. Conceituar os operadores & e;
3. Manipular ponteiro com os operadores & e;
4. Compreender o uso do operador ->;
5. Conceituar alocação dinâmica de memória;
6. Compreender o uso de ponteiro no estudo de Listas Lineares;
7. Conceituar listas lineares simplesmente encadeadas;
8. Representar listas lineares simplesmente encadeadas com poucos nodos.

1 Alocação Dinâmica

Durante todo esse tempo que estamos estudando Algoritmos e Estruturas de Dados, temos determinado quanto de memória nossos programas vão usar através das declarações que fazemos das variáveis locais, globais, matrizes, sejam elas de tipos primitivos ou de structs.

Muitas vezes, sentimos necessidade de determinar durante a execução do programa a quantidade de elementos de uma matriz unidimensional, por exemplo, mas, ou nós superdimensionávamos alocando mais do que iria ser usado para que o usuário pudesse entrar com “qualquer valor(?)”, ou não conseguíamos fazer o que queríamos.

A alocação dinâmica de memória vem suprir essa deficiência e possibilitar a criação de tipos de dados e estruturas de qualquer tamanho durante a execução do programa.

Sabemos que durante a execução de um programa, o Sistema Operacional, aloca espaço na Memória Principal para as variáveis locais e globais, outro espaço é alocado para armazenar o código do programa, outro espaço para armazenamento das funções e o que sobra, é chamado de Área de memória Livre(free store) ou Heap que é usada para alocação dinâmica de memória. É bom frisar que a área da Pilha cresce em direção ao Heap e o Heap, em direção à área da Pilha.



A figura abaixo representa essas quatro regiões da memória principal e não significa que sempre será assim como afirma Schildt, H(1996, p.14):



“Embora a disposição física exata de cada uma das quatro regiões possa diferir entre tipos de CPU e implementações...”

Operadores *new* e *delete*

Na linguagem C++, alocar dinamicamente é relativamente simples porque temos somente dois operadores para alocar e desalocar memória.

Entretanto, é imprescindível o conhecimento sobre ponteiros.

No momento, só falaremos das funções desses operadores porque como usá-los, só depois que estudarmos ponteiros.

Saiba mais



Clique no link a seguir e saiba mais sobre os operadores *new* e *delete*:

http://estaciodocente.webaula.com.br/cursos/gon119/docs/08ED_DOC01.pdf

Defendo a teoria de que não se pode saber superficialmente um conceito que permeia a alocação dinâmica de memória.

Não seremos só programadores e, sendo assim, temos que entender o que se passa na memória principal para que possamos desenvolver códigos sem problemas e mais eficientes. Se isso não fosse verdade, nos nossos cursos não teríamos disciplinas tais como Organização de Computadores, Sistemas Operacionais entre outras.

Não posso me aprofundar no estudo de ponteiros, mas vou tentar explicar de uma forma simples para que você possa caminhar sozinho, se assim desejar.

A MP consiste de trilhões de posições para armazenamento do Sistema Operacional, dados, programas, etc. Seu tamanho depende de quanto sua placa mãe suporta porque se foi o tempo em que o dinheiro era o principal fator para se aumentar a memória, visto que hoje em dia ela está relativamente barata.

Nós já vimos que, ao declararmos uma variável nas linguagens de alto nível e de nível intermediário, o "nome" dado é associado a uma posição de memória (endereço). Este tipo de variável que nós conhecemos, armazena dado.

Hoje vamos conhecer a variável ponteiro. Esta é uma variável que armazena o endereço de outra variável, possibilitando receber o endereço que é retornado quando se aloca a memória dinamicamente.

Atenção

Uma boa dica para entender como funciona a variável ponteiro.

Você leu no jornal um anúncio de uma loja que vende notebook, muito facilitado, para alunos que estão cursando ADS ou Sistemas de Informação.

Ao ligar para a loja, você pede o endereço e anota na sua agenda. Então, sua agenda aponta para a loja, visto que está anotado o endereço da loja (&loja).

Para dar uma olhada em outros modelos, você resolve ir até a loja(&loja) e chegando lá, viu o que estava dentro da loja(*loja).

2 Declaração da variável ponteiro

```
tipo *nomeDaVariávelPonteiro ;
```

O que significa tipo para uma variável ponteiro?

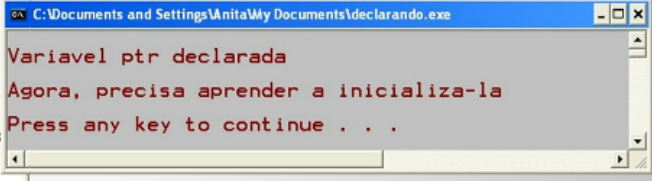
Quando declaramos uma variável ponteiro, precisamos informar o tipo de variável que será apontada pelo ponteiro, isto é, char, int, float ou double.

```
int *ptrAno;  
float *ptrNota;  
char *ptrNome;
```

Por que temos que usar o * antes do nome da variável ponteiro?

Porque o asterisco é o caracter(alguns autores chamam de operador) que sinaliza que a variável está sendo declarada como ponteiro.

```
#include <iostream>  
using namespace std;  
main()  
{  
    float *ptr;  
    cout<<"\nVariavel ptr declarada ";  
    cout<<"\n\nAgora, precisa aprender a inicializa-la";  
    cout<<"\n\n";  
    cout<<"\n\n";  
    system("pause");  
}
```



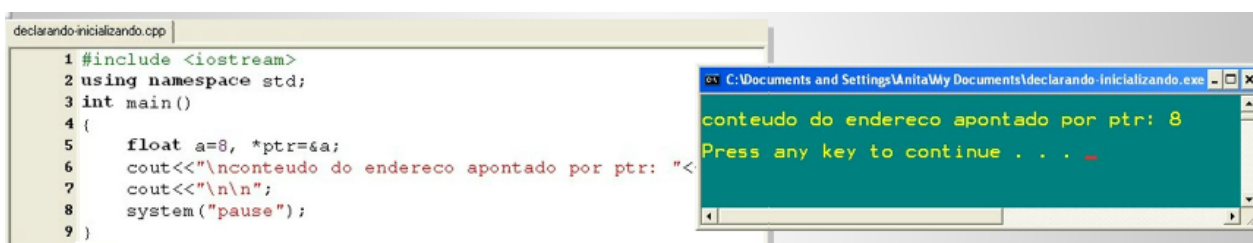
Qual a finalidade do caracter & antes de uma variável?

O operador & será usado antes de uma variável quando quisermos o endereço da variável e não, seu conteúdo.

Como se inicializa uma variável ponteiro?

A inicialização poderá acontecer na declaração da mesma forma que fazemos com as variáveis simples ou depois da declaração através de um comando de atribuição onde a variável ponteiro receberá o endereço da variável que ela irá apontar.

tipo nomeDaVariável=valor, *nomeDaVariávelPonteiro = &nomeDaVariável;



The screenshot shows a C++ IDE with a file named 'declarando-inicializando.cpp'. The code is as follows:

```
1 #include <iostream>
2 using namespace std;
3 int main()
4 {
5     float a=8, *ptr=&a;
6     cout<<"\nconteudo do endereco apontado por ptr: "<
7     cout<<"\n\n";
8     system("pause");
9 }
```

Next to the code is a console window titled 'C:\Documents and Settings\Anita\My Documents\declarando-inicializando.exe'. It displays the output of the program:

```
conteudo do endereco apontado por ptr: 8
Press any key to continue . . .
```

Atenção

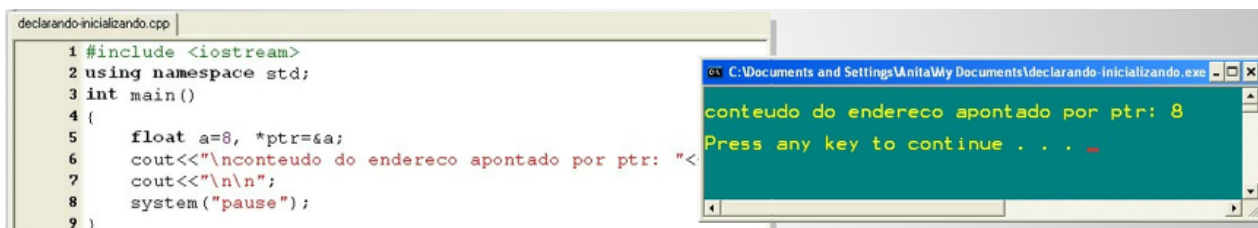
No exemplo anterior, inicializamos a variável ponteiro na declaração. Clique no link a seguir e veja um exemplo após a declaração.

http://estaciODOcente.webaula.com.br/cursos/gon119/docs/08ED_DOC02.pdf

nomeDaVariávelPonteiro = &nomeDaVariável;

O asterisco será usado em outra ocasião quando trabalharmos com variável ponteiro?

Sim, quando desejarmos saber o conteúdo do endereço apontado por uma variável ponteiro.



The image shows a C++ IDE with two windows. The left window, titled 'declarando-inicializando.cpp', contains the following code:

```
1 #include <iostream>
2 using namespace std;
3 int main()
4 {
5     float a=8, *ptr=&a;
6     cout<<"\nconteudo do endereco apontado por ptr: "<
7     cout<<"\n\n";
8     system("pause");
9 }
```

The right window, titled 'C:\Documents and Settings\AnitaW\Documents\declarando-inicializando.exe', shows the output of the program:

```
conteudo do endereco apontado por ptr: 8
Press any key to continue . . .
```

Atenção

No exemplo anterior, inicializamos a variável ponteiro na declaração. Clique no link a seguir e veja um exemplo após a declaração.

http://estaciadocente.webaula.com.br/cursos/gon119/docs/08ED_DOC02.pdf

3 Ponteiros para Estruturas

Nada nos impede de passarmos uma estrutura para uma função, entretanto isso sobrecarrega a pilha, aumentando o tempo de transferência dos dados entre as funções.

Para minimizar esse problema e tendo em vista que já estudamos funções, vamos aprender agora como passar somente um ponteiro como argumento para função, e, através dele, o endereço da estrutura.

Saibam também que será de muita utilidade nas próximas aulas.

Existem duas formas de se acessar o membro através do ponteiro. Usando o operador -> ou o *.



Para se atribuir o endereço de uma estrutura a um ponteiro, usamos um comando de atribuição:



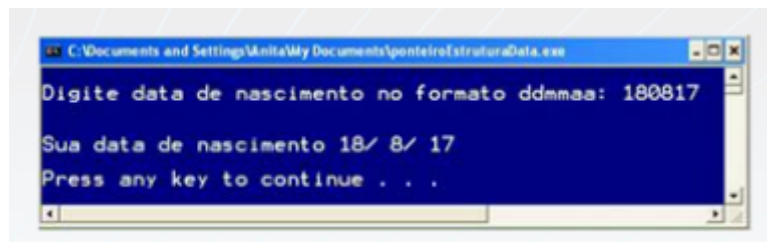
Vamos começar com um bem simples e um dos meus preferidos.

Observe o código e a saída.

```

1 #include <iostream>
2 #include <cstring>
3 using namespace std;
4
5 struct data
6 {
7     int dia, mes, ano;
8 };
9
10 int main()
11 {
12     data dataNasc, *ptr;
13     int d;
14     cout<<"\nDigite data de nascimento no formato ddmmaa: ";
15     cin>>d;
16     ptr=&dataNasc;
17     ptr->dia=d/10000;
18     ptr->mes=(d/100) % 100;
19     ptr->ano=d%100;
20     cout<<"\n\nSua data de nascimento "<<ptr->dia<<"/ " <<ptr->mes<<"/ " <<ptr->ano;
21     cout<<"\n\n"; system("pause");
22 }

```



Análise do código

```

1 #include <iostream>
2 #include <cstring>
3 using namespace std;
4
5 struct data
6 {
7     int dia, mes, ano;
8 };
9
10 int main()
11 {
12     data dataNasc, *ptr;
13     int d;
14     cout<<"\nDigite data de nascimento no formato ddmmaa: ";
15     cin>>d;
16     ptr=&dataNasc;
17     ptr->dia=d/10000;
18     ptr->mes=(d/100) % 100;
19     ptr->ano=d%100;
20     cout<<"\n\nSua data de nascimento "<<ptr->dia<<"/ " <<ptr->mes<<"/ " <<ptr->ano;
21     cout<<"\n\n"; system("pause");
22 }

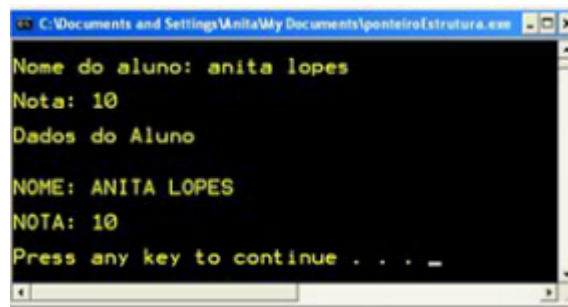
```


- 1) Na linha 5, foi criada uma estrutura global de nome data com três membros: dia, mes e ano.
- 2) Na linha 12, foram declaradas duas variáveis do tipo data. Uma de nome data dataNasc e outra, ponteiro, de nome ptr.
- 3) Na linha 16, a variável ptr recebeu o endereço da variável dataNasc.
- 4) Usei a notação da seta para acessar os membros.

Neste próximo exemplo, além do que já foi visto no anterior, passaremos um ponteiro para função.

Observe o código e a saída

```
1 #include <iostream>
2 #include <cstring>
3 using namespace std;
4
5 struct CADASTRO
6 {
7     char nome[30];
8     float nota;
9 };
10
11 void exibe(CADASTRO *m);
12 int main()
13 {
14     CADASTRO aluno,*p;
15     cout<<"\nNome do aluno: "; cin.getline(aluno.nome,30);
16     cout<<"\nNota: "; cin>>aluno.nota;
17     p=&aluno; //aponta para a estrutura
18     exibe(p);
19     cout<<"\n\n"; system("pause");
20 }
21
22 void exibe(CADASTRO *m)
23 {
24     int c;
25     for (c=0;c<strlen(m->nome);c++)
26         m->nome[c]=toupper(m->nome[c]);
27     cout<<"\nDados do Aluno\n";
28     cout<<"\n\nNOME: "<<m->nome; //usei as duas formas para vocês verem
29     cout<<"\n\nNOTA: "<<(*m).nota;
30 }
```



```
C:\Documents and Settings\Anita\My Documents\ponteiroEstrutura.exe
Nome do aluno: anita lopes
Nota: 10
Dados do Aluno
NOME: ANITA LOPES
NOTA: 10
Press any key to continue . . . _
```

Análise do código

```

1 #include <iostream>
2 #include <string>
3 using namespace std;
4
5 struct CADASTRO
6 {
7     char nome[30];
8     float nota;
9 };
10
11 void exibe(CADASTRO *m);
12 int main()
13 {
14     CADASTRO aluno, *p;
15     cout<<"\nNome do aluno: "; cin.getline(aluno.nome, 30);
16     cout<<"\nNota: "; cin>>aluno.nota;
17     p=&aluno; //aponta para a estrutura
18     exibe(p);
19     cout<<"\n\n"; system("pause");
20 }
21
22 void exibe(CADASTRO *m)
23 {
24     int c;
25     for(c=0; c<strlen(m->nome); c++)
26         m->nome[c]=toupper(m->nome[c]);
27     cout<<"\nDados do Aluno\n";
28     cout<<"\nNOME: " <<m->nome; //usei as duas formas para vocês verem
29     cout<<"\nNOTA: " <<(*m).nota;
30 }

```

- 1) Na linha 11, temos o protótipo da função: void exibe(CADASTRO *m);
É uma função sem retorno, de nome exibe e que tem como parâmetro uma variável ponteiro que aponta para uma estrutura de gabarito(identificador) CADASTRO.
- 2) Na linha 14, declara duas variáveis do tipo CADASTRO, sendo uma ponteiro(p) e a outra, aluno.
- 3) Na linha 17, a variável p é inicializada com endereço da estruturas: p=&aluno;.
- 4) Nas linhas 28 e 29, mostrei as duas formas de acessar o membro de uma estrutura.

4 Ponteiro e Alocação Dinâmica

No início desta aula, vimos que a alocação dinâmica da memória na linguagem C++ é feita pelo operador new, mas como é retornado o primeiro endereço do bloco alocado, precisamos armazenar esse endereço e é aí que está a importância, no processo, da variável ponteiro.

Declarando e Inicializando um ponteiro para alocar, dinamicamente, a memória.

```

tipo *nomeDoPonteiro = new tipo;
tipo *nomeDoPonteiro = new tipo[...];

```

Vamos entender cada parte destas duas sintaxes.

1ª sintaxe

- tipo – é o tipo do ponteiro.
- * - operador que sinaliza que a variável é um
- ponteiro.new – aloca a memória dinamicamente.
- tipo – é o tipo do dado, compatível com o tipo do ponteiro.

2ª sintaxe

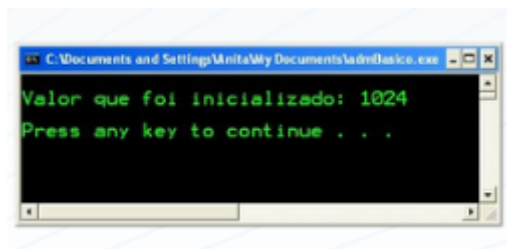
- tipo – é o tipo do ponteiro.
- * - operador que sinaliza que a variável é um
- ponteiro.new – aloca a memória dinamicamente.
- tipo[...] – é o tipo do dado, compatível com o tipo do ponteiro. Aloca uma quantidade especificada dentro dos colchetes.

Exemplos

<code>float *ptF= new float;</code>	Declara e inicializa um ponteiro que aponta para um lugar que armazena dado real. O ponteiro armazenará o primeiro endereço do bloco que armazena dado real.
<code>int *ptI= new int(25);</code>	Declara e atribui o valor 25 ao endereço apontado por ptI.
<code>int *pt=new int[50];</code>	Declara e inicializa um ponteiro que aponta para array. O ponteiro armazenará o primeiro endereço do bloco que armazena dados inteiros.

A primeira alocação dinâmica

```
admBasico.cpp
1 #include <iostream>
2 using namespace std;
3 int main()
4 {
5     int *pt = new int(1024);
6     cout<<"\nValor que foi inicializado: "<<*pt<<"\n\n";
7     system("pause");
8 }
```



Atenção

Clique no link a seguir e saiba como realizar um teste para saber da possibilidade de alocar memória:

http://estaciodocente.webaula.com.br/cursos/gon119/docs/08ED_DOC03.pdf

5 Ponteiro e matriz de estruturas

A relação entre matrizes e ponteiros é muito grande e, infelizmente, não temos como nos aprofundar nos assuntos. Entretanto, como vamos trabalhar com estruturas, resolvi apresentar este exemplo de matrizes de estruturas com alocação dinâmica porque vou mostrar como acessar um elemento da matriz usando o ponteiro com deslocamento. Atenção para os passos.

Observe a struct e a linha de alocação dinâmica.



I. ptr é um ponteiro que aponta para uma matriz de struct.

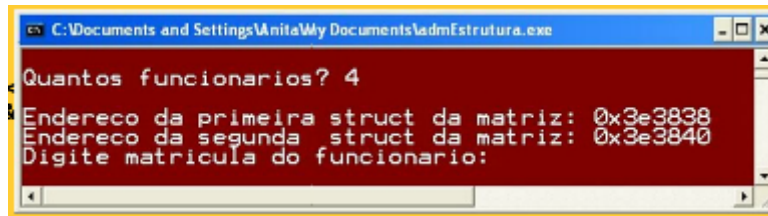
II. ptr + x (considere x como sendo uma variável do for). Quando um ponteiro se desloca de um, na verdade, ele não se desloca para o

endereço vizinho, ele avança tantos endereços quantos forem os bytes da variável/estrutura apontada por ele que neste caso, são 8. Se você incluísse, depois da linha 16, duas linhas que exibissem esses endereços, obteríamos a saída abaixo.

```
16 dados *ptr = new dados[nfunc];
```

```
count<<"\nEndereco da primeira struct da matriz:<<ptr[0];
```

```
count<<"\nEndereco da segunda struct da matriz:<<ptr[1];
```



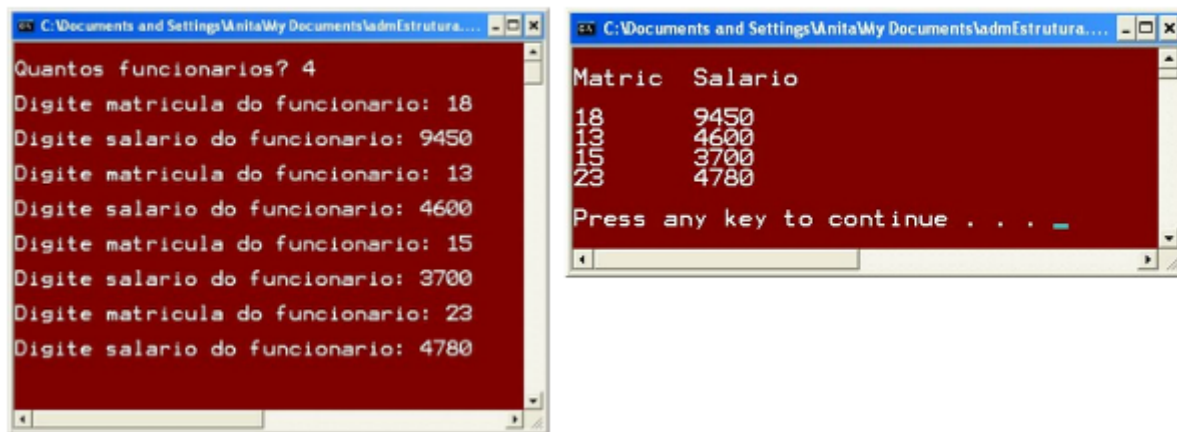
Observe que de 0x 3e3838 ate 0x3e3840, endereços em hexadecimal, temos os seguintes endereços: 3e3838, 3e3839, 3e383A, 3e383B, 3e383C, 3e383D, 3e383E e 3e383F, totalizando 8 endereços (4 endereços para int e 4 endereços para float).

III. (ptr+x) -> matric é equivalente à ptr[x].matric porque você pode tratar o ponteiro como se fosse uma matriz e amatriz, como se fosse um ponteiro:

Observe o código e a saída.



```
1 #include <iostream>
2 #include <cstring>
3 using namespace std;
4
5 struct dados
6 {
7     int matric;
8     float sal;
9 };
10
11 int main()
12 {
13     int x,nfunc;
14     cout<<"\nQuantos funcionarios? ";
15     cin>>nfunc;
16     dados *ptr= new dados[nfunc];
17
18     for(x=0; x<nfunc; x++)
19     {
20         cout<<"\nDigite matricula do funcionario: ";
21         cin>>ptr[x].matric;
22         cout<<"\nDigite salario do funcionario: ";
23         cin>>ptr[x].sal;
24     }
25     system("cls");
26     cout<<"\nMatric\tSalario\n";
27     for(x=0; x<nfunc; x++)
28         cout<<"\n"<<(ptr+x)->matric<<"\t"<<(ptr+x)->sal;
29     cout<<"\n\n"; system("pause");
30 }
```



Nosso estudo sobre ponteiros procurou lhe dar embasamento para que você possa começar seu estudo sobre as Estruturas de Dados Dinâmicas.

Não teve como objetivo se aprofundar na aritmética dos ponteiros, ponteiros para funções, matrizes de ponteiros, ponteiros para ponteiros, etc.

Como vamos usar esses conceitos muitas vezes, creio que acabaremos dominando com muita facilidade.

Vamos para a segunda parte da aula!!!

6 Listas encadeadas

Nas aulas 5, 6 e 7 tivemos oportunidade de estudar as estruturas de dados lineares Lista, Pilhas e Filas com seus elementos armazenados na memória de forma contígua e estática.

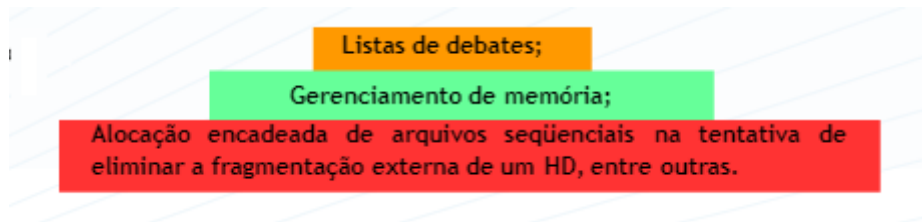
Aprendemos também que as inserções e remoções nas Pilhas e nas Filas acontecem em extremidades determinadas enquanto que nas Listas, não existem regras sobre isso.

Na aula de hoje, começaremos a estudar as estruturas dinâmicas e, por essa razão, tivemos que conhecer a variável ponteiro porque sem ela, seria impossível implementar a alocação dinâmica, visto que o operador new, retorna, se bem sucedida a alocação, o primeiro endereço do bloco que precisa ficar armazenado. Em resumo, sem fazer trocadilho, tudo muito bem encadeado: ponteiro -> alocação dinâmica -> Listas Encadeadas.

Listas de encadeamento simples, normalmente chamadas simplesmente de Listas Encadeadas, são compostas de elementos individuais, cada um ligado por um único ponteiro. Cada elemento consiste de duas partes: um membro de dados e um ponteiro.(LOUDON, K, 2000, p.56)

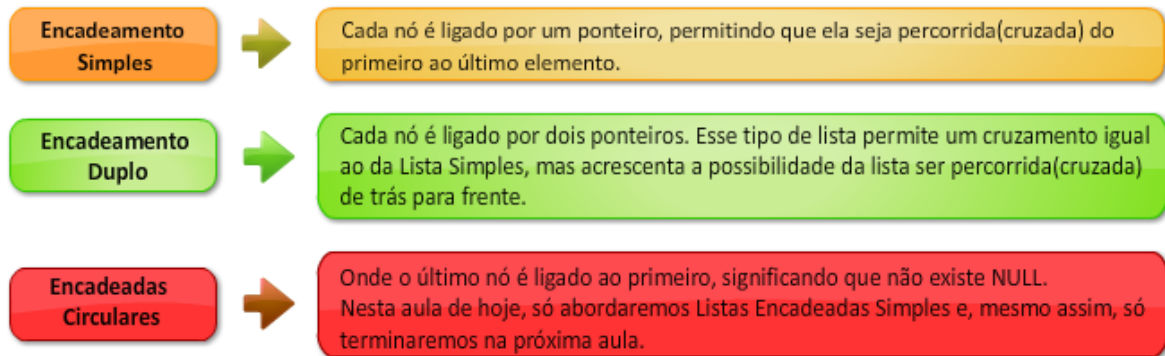
As Listas Encadeadas, também conhecidas como listas ligadas, são consideradas uma das mais importantes estruturas de dados. Talvez a forma livre de inserir e remover elementos em qualquer posição e as aplicações importantes que fazem uso dela tenham lhe dado esse status.

Podemos enumerar algumas dessas aplicações:

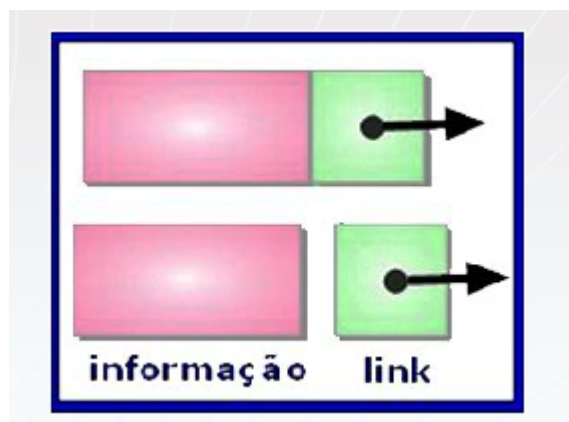


As Listas Encadeadas são estruturas dinâmicas e, por essa razão, não existe restrição, a não ser de memória, de aumentar, ou diminuir, de tamanho durante a execução do programa.

As Listas Encadeadas podem ser:



As Listas Encadeadas Simples são formadas por nós que têm o seguinte o formato:



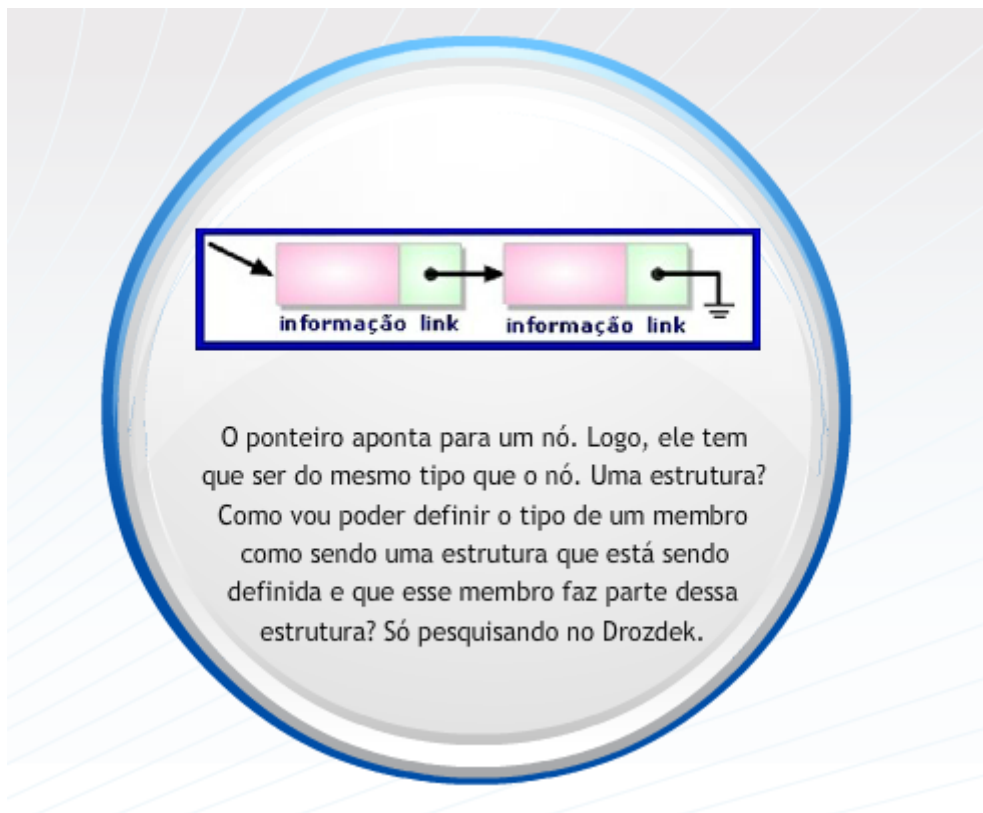
Pela figura, podemos observar que cada nó tem duas partes. Uma, armazena a informação e a outra, o endereço do próximo

Se olharmos para este nó, podemos entendê-lo como uma estrutura, visto que ele é formado por dois tipos diferentes, sendo um, necessariamente um ponteiro.

Se formos audaciosos, diríamos que é uma variável struct com dois membros e, para chegarmos mais próximos do nó da Lista, poderíamos nomear esta estrutura de no.

O tipo do membro que armazena a informação não será difícil de identificarmos porque dependerá da aplicação, mas qual será o tipo da variável ponteiro?

No nosso breve estudo sobre ponteiros, aprendemos que o tipo do ponteiro tem que ser do mesmo tipo que o elemento que ele vai apontar.



Concordo que é meio complexo esse questionamento, mas se você fizer algumas leituras desse parágrafo, vais acabar entendendo que não tinha alternativa já que o ponteiro tinha que ser do mesmo tipo que o nó.

Acredito que a única dificuldade ficará por conta de permitir que esta definição possa acontecer dentro dela mesma. Como DROZDEK, A(2002,p.68) diz: “Essa circularidade, no entanto, é permitida em C++”.

Agora, vamos construir a struct.


```
struct no
{
    <tipo> info1;
    <tipo> info2;
    struct no* link;
};
```



Pensei que só poderia ter um membro fora o ponteiro. Agora você diz que podem ser vários?



Fique tranqüila. Você já vem treinando struct desde a Aula 3.

Operações que podem ser realizadas com as Listas Simplesmente Encadeadas

Inicializar

O processo de inicialização de uma Lista consiste em criar uma Lista vazia, isto é uma lista sem elementos.

A Lista sempre será representada pelo ponteiro para o primeiro elemento e, se é uma lista vazia, seu valor será NULL(constante escrita com letras maiúsculas).

Você pode inicializar a Lista na declaração.

Exemplo: no *Lista = NULL;

Inserir um nó no início ou no fim ou em qualquer posição

Após a criação da Lista, os nós podem ser inseridos, um a um, ou aos grupos, dependendo como você implementou seu código.

Como não existe restrição para posição de inserção na Lista, você vai escolher, mas você concluirá que a forma mais simples é a de inserir no início da Lista.

Remover nó

A remoção de um nó poderá ser mais simples, ou mais complexa. Tudo vai depender da sua posição na Lista.

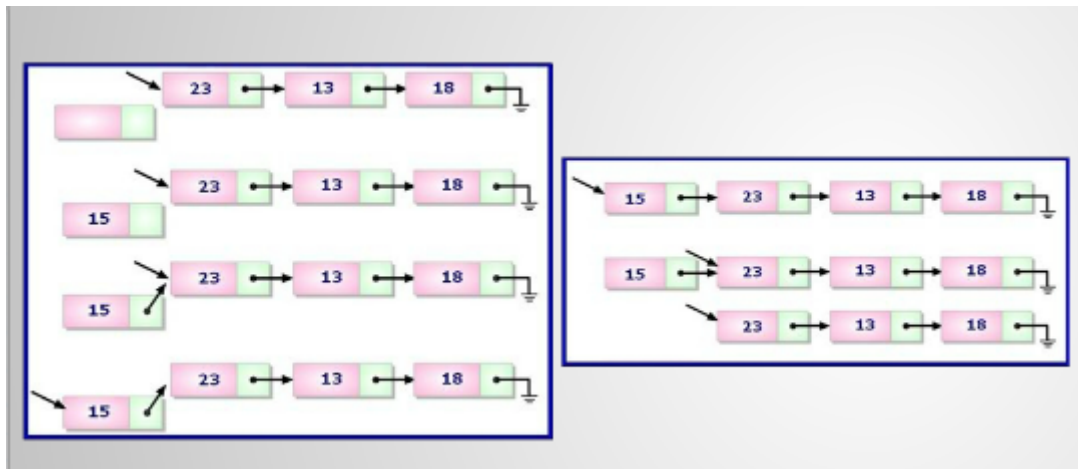
Insere nó na frente, em qualquer lugar ou insere nó atrás?

Remove nó que está na frente, em qualquer lugar ou nó que está atrás?

Quem lê pensa que estamos falando ora de Fila, ora de Pilha ou ora, de Lista já que não tem restrição todas.

Claro que para implementar algumas formas são mais simples do que outras. Entretanto vamos tentar ensinar apresentar as três formas.

Observe as figuras abaixo onde nós são inseridos e removidos e, depois de analisar, pense qual estrutura poderia ser essa, tendo onde os dados são inseridos/removidos e quais seriam as etapas da inserção e da remoção de um nó?



Depois que escrever, clique no botão solução.

Lembre-se de que não precisa estar com as mesmas palavras. O que vale é o sentido.

Solução

Resposta

Como Insere na frente e Remove na frente, é a mesma filosofia da PILHA.

Etapas para inserir um nó no início da Lista

- 1 - Foi criado um novo nó;
- 2 - O novo nó foi inicializado com valor 15;
- 3 - Como o novo nó foi inserido na frente, o membro que contém o apontador deste novo nó se torna um ponteiro para o primeiro nó da lista atual;
- 4 - O novo nó se torna o primeiro da lista e, sendo assim, o ponteiro que sinaliza o primeiro nó, é desviado para o nó incluído.

Etapas para remover um nó no início da Lista

- 1) O ponteiro que sinaliza o primeiro nó da Lista é reajustado para que o segundo nó passe a ser o primeiro.
- 2) Algo que você não poderia saber: Para que seja possível a etapa 1, a informação do nó removido é armazenada para que depois de finalizar a etapa 1, possa ser deletada(delete).

Percorrer a lista

Percorrer a lista significa “passar” por todos os nós. Seja para exibir todos os nós, contar os nós, buscar um elemento para remoção ou para alterar seu valor.

Além destas, ainda temos as seguintes operações:

- EXIBIR A LISTA;
- DETERMINAR O NÚMERO DE NÓS;
- LOCALIZAR UM NÓ;
- ALTERAR O CONTEÚDO DE UM NÓ, ENTRE OUTRAS.

Mas, não vou assustá-lo. Vamos dar um passo de cada vez para que possamos aprender tudo. Afinal, essa parte da matéria é mais abstrata do que todas as outras e, se só escrevermos muito conteúdo, creio que não ficará bem fixado. Por essa razão, optei por um tipo de aula diferente das demais, isto é, explica e exemplifica.

Para que possamos fazer o passo a passo, vamos supor que nosso nó tenha como membro um número inteiro fora o ponteiro. Assim, vamos construir uma lista com um nó.

Definindo a struct

```
struct nodo
{
    int num;
    struct nodo* prox;
};
```

Criando nó

```
nodo* no1=new nodo;
```

Atribuindo valores aos membros

```
no1->num=23;
no1->prox=NULL;
```

Exibindo

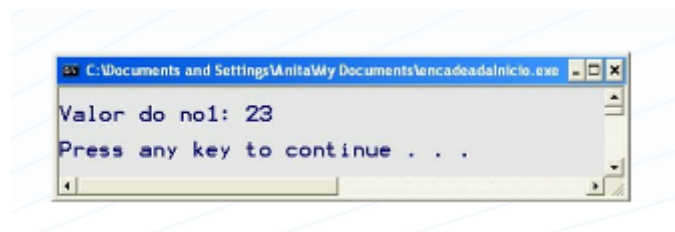
```
cout<<"\nValor do no1: "<<no1->num;
```

Liberando

```
delete no1;  
no1=0; //pelo que falamos
```

Construindo o código e exibindo resultado

```
encadeadaInicio.cpp  
1 #include <iostream>  
2 #include <cstdlib>  
3 using namespace std;  
4 int main()  
5 {  
6     struct nodo  
7     {  
8         int num;  
9         struct nodo* prox;  
10    };  
11  
12    //criando no  
13    nodo* no1=new nodo;  
14    //atribuindo valor ao elemento  
15    no1->num=23;  
16    // atribuindo 0 ou NULL ao campo proxima celula  
17    no1->prox=NULL;  
18    //exibindo  
19    cout<<"\nValor do no1: "<<no1->num;  
20    //liberando  
21    delete no1; no1=0;  
22    cout<<"\n\n";  
23    system("pause");  
24 }
```

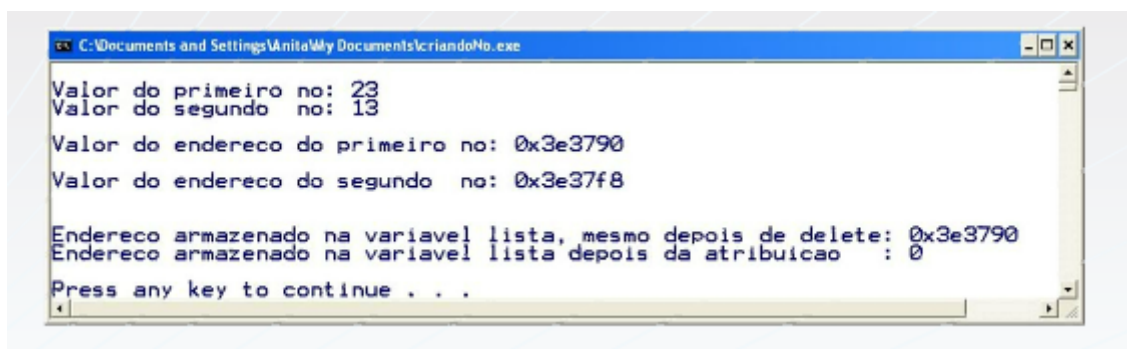


Agora que conseguimos, vamos construir uma lista com dois nós. Usaremos a mesma struct para facilitar. Observe que os dois nós serão dependentes da mesma variável ponteiro.

```

1 #include <iostream>
2 #include <cstdlib>
3 using namespace std;
4 int main()
5 {
6
7     struct nodo
8     {
9         int num;
10        struct nodo* prox;
11    };
12
13    nodo* lista=new nodo; //criando primeiro no
14    lista->num=23;
15    lista->prox=NULL;
16
17    lista->prox=new nodo; //criando segundo no
18    lista->prox->num=13;
19    lista->prox->prox=NULL;
20
21    //exibindo
22    cout<<"\nValor do primeiro no: "<<lista->num;
23    cout<<"\nValor do segundo no: "<<lista->prox->num;
24    cout<<"\n\nValor do endereco do primeiro no: "<<lista;
25    cout<<"\n\nValor do endereco do segundo no: "<<lista->prox;
26    //liberando
27    delete lista;
28    cout<<"\n\n\nEndereco armazenado na variavel lista, mesmo depois de delete: "<<lista;
29    lista=0;
30    cout<<"\n\n\nEndereco armazenado na variavel lista depois da atribuicao : "<<lista;
31    cout<<"\n\n";
32    system("pause");
33 }

```



1 O trecho abaixo cria e atribui valores aos membros do struct, lembrando que o membro

próximo é um ponteiro que irá receber o valor NULL(pode ser 0), supondo que não tem mais elemento na lista.

```
13 nodo* lista=new nodo;
```

```
14 lista->num=23;
```

```
15 lista->prox=NULL;
```

2 O trecho abaixo cria o segundo nó, mas necessita de uma explicação para a criação do segundo nó. Se lista->prox tinha recebido NULL, significava que era o último nó da Lista. Sendo assim, esta variável ponteiro recebeu o endereço retomado pelo operador new quando ele alocou um bloco na memória, dinamicamente. O "nome" então desse nó passou a ser lista->prox e, por essa razão, que os dois membros estão sendo referenciados precedido por esse "nome". 17 lista->prox=new nodo; //criando segundo no 18 lista->prox->num=13; 19 lista->prox->prox=NULL;

3 O trecho abaixo exibe na tela os valores armazenados nos membros num e os endereços de cada nó.

Reforçando que o endereço que retorna da alocação é o primeiro do bloco logo, é o endereço do primeiro nó e como o membro ponteiro aponta para o outro nó, para sabermos o endereço do segundo nó, chamamos pelo primeiro membro ponteiro do primeiro nó.

```
21 cout<<"\nValor do primeiro no: "<<lista->num;
```

```
22 cout<<"\nValor do segundo no: "<<lista->prox->num;
```

```
23 cout<<"\n\nValor do endereco do primeiro no: "<<lista;
```

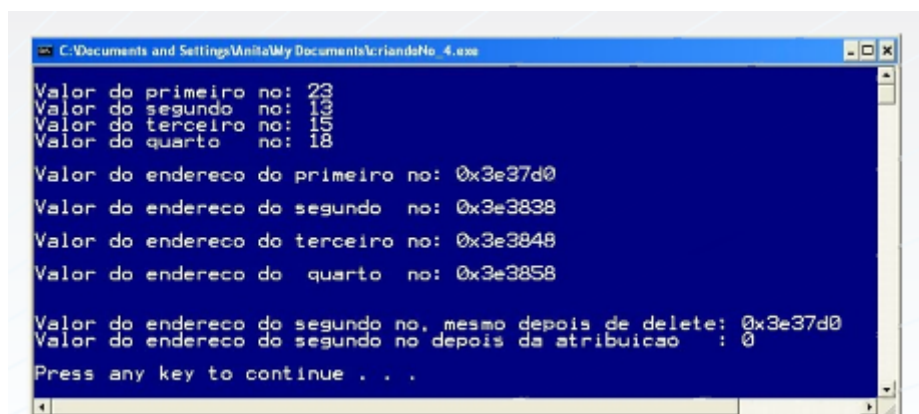
```
24 cout<<"\n\nValor do endereco do segundo no: "<<lista->prox;
```

Acabamos de analisar um código com dois nós. Que tal você aumentar esse código para quatro nós? Construa seu código é só clique no botão solução para comparar. Lembre-se de que se não construir, não se aprende só olhando, ou copiando.

```

1 #include <iostream>
2 #include <cstdlib>
3 using namespace std;
4 int main()
5 {
6
7     struct nodo
8     {
9         int num;
10         struct nodo* prox;
11     };
12
13     //primeiro nó
14     nodo* lista=new nodo;
15     lista->num=23;
16     lista->prox=NULL;
17     //segundo nó
18     lista->prox=new nodo;
19     lista->prox->num=13;
20     lista->prox->prox=NULL;
21     //terceiro nó
22     lista->prox->prox=new nodo;
23     lista->prox->prox->num=15;
24     lista->prox->prox->prox=NULL;
25     //quarto nó
26     lista->prox->prox->prox=new nodo;
27     lista->prox->prox->prox->num=18;
28     lista->prox->prox->prox->prox=NULL;
29     cout<<"\nValor do primeiro no: "<<lista->num;
30     cout<<"\nValor do segundo no: "<<lista->prox->num;
31     cout<<"\nValor do terceiro no: "<<lista->prox->prox->num;
32     cout<<"\nValor do quarto no: "<<lista->prox->prox->prox->num;
33     cout<<"\n\nValor do endereço do primeiro no: "<<lista;
34     cout<<"\n\nValor do endereço do segundo no: "<<lista->prox;
35     cout<<"\n\nValor do endereço do terceiro no: "<<lista->prox->prox;
36     cout<<"\n\nValor do endereço do quarto no: "<<lista->prox->prox->prox;
37     //liberando
38     delete lista;
39     cout<<"\n\nValor do endereço do segundo no, mesmo depois de delete: "<<lista;
40     lista=0;
41     cout<<"\n\nValor do endereço do segundo no depois da atribuição : "<<lista;
42     cout<<"\n\n";
43     system("pause");
44 }

```



```

C:\Documents and Settings\Anita\My Documents\criandoNo_4.exe

Valor do primeiro no: 23
Valor do segundo no: 13
Valor do terceiro no: 15
Valor do quarto no: 18

Valor do endereço do primeiro no: 0x3e37d0
Valor do endereço do segundo no: 0x3e3838
Valor do endereço do terceiro no: 0x3e3848
Valor do endereço do quarto no: 0x3e3858

Valor do endereço do segundo no, mesmo depois de delete: 0x3e37d0
Valor do endereço do segundo no depois da atribuição : 0
Press any key to continue . . .

```

Atenção

Sei que nem tudo pode ser abordado em uma aula, mas vou apresentar agora como se insere um nó na frente de uma Lista e como se insere um nó atrás de uma Lista. Não usaremos ainda função.

Inserir nó ao final

Preferi usar dois nós com nomes diferentes, para facilitar.

```
//criando no1
nodo* no1=new nodo;
no1->num=23;
no1->prox=NULL;

//criando no2
nodo* no2= new nodo;
no1->prox=no2;
no2->num = 13;
no2->prox = NULL;
```

A inserção ao final com dois nós de nomes diferentes também é de fácil entendimento e a única linha que poderia causar certa estranheza é a que atribui o endereço **de no2 ao membro ponteiro de no1**, mas como precisa ter encadeamento e vamos inserir depois do **no1** fica evidente que o **membro ponteiro de no1** precisa apontar para o novo nó.

Observe o código e a saída

```
encadeadodinsereFinal.cpp
1 #include <iostream>
2 #include <cstdlib>
3 using namespace std;
4 int main()
5 {
6     struct nodo
7     {
8         int num;
9         struct nodo* prox;
10    };
11    //criando no1
12    nodo* no1=new nodo;
13    no1->num=23;
14    no1->prox=NULL;
15    //criando no2
16    nodo* no2= new nodo;
17    no1->prox=no2;
18    no2->num = 13;
19    no2->prox = NULL;
20    //exibindo
21    cout<<"\nValor de no1: "<<no1->num;
22    cout<<"\nValor de no2: "<<no2->num;
23    cout<<"\n\nEndereco de no1: "<<no1;
24    cout<<"\nEndereco de no2: "<<no2;
25    cout<<"\n\nEndereco apontado por no1: "<<no1->prox;
26    cout<<"\nEndereco apontado por no2: "<<no2->prox;
27    //liberando
28    delete no1; no1=0; delete no2; no2=0;
29    cout<<"\n\n";
30    system("pause");
31 }
```

Análise do código

Coloquei para exibir os endereços de no1 e no2 para que você pudesse constatar que no2 for colocado atrás de no1 na lista porque no1->prox aponta para o endereço de no2.


```
C:\Documents and Settings\Unita\My Documents\Encadeada\lmaereFinal....
Valor do no1: 23
Valor do no2: 13

Endereco de no1: 0x3e2798
Endereco de no2: 0x3e3800

Endereco apontado por no1: 0x3e3800
Endereco apontado por no2: 0
Press any key to continue . . .
```

Acredito que já tivemos oportunidade de entendermos o básico de uma Lista Encadeada e que estamos prontos para a 2a etapa.

Nesta etapa, os códigos não estão com funções porque espero que vocês as construam até a próxima aula, visto que apresentarei três programas que neles já estão presentes os trechos que deverão se transformar em funções. Para dar uma ajuda maior, darei destaque desses trechos para vocês.



Escolhi os trechos de inserção de nó na Frente da Lista, listar e remover nó no Início da Lista para começar.

Na próxima aula, construiremos outros, mas fique à vontade para pesquisar.

1) Trecho principal do programa -Insere um nó na Frente da lista.

```
//inserir nó na Frente
temp = new nodo;
temp->num = 23;
temp->prox=lista ;
lista= temp;
```

```
C:\Documents and Settings\Unita\My Documents\criando lista.exe
Inserindo No na Frente - Listando um a um
Valor do 1o no: 18
Valor do 2o no: 15
Valor do 3o no: 13
Valor do 4o no: 23
Press any key to continue . . .
```

Análise do código

l) Linha 11

```
nodo *temp, *lista=NULL;
```

Declara temp que servirá para receber o novo nó e lista que é inicializada, visto que recebeu NULL. Já vimos que podemos criar uma lista vazia.

II) Linhas 14-17 - Atenção para esse trecho porque ele irá se transformar em uma função.

```
13 temp = new nodo; //aloca dinamicamente um nó do tipo nodo
```

```
14 temp->num = 23; // atribui 23 ao membro num
```

```
15 temp->prox=lista ; // atribui o endereço de lista ao membro ponteiro prox
```

```
16 lista= temp; // atualiza lista
```

Saiba mais



Que tal você dar uma olhada na figura do exercício de inserir/ remover nó que pedi para que você relacionasse as etapas e comparasse com este trecho.

```

criandoLista.cpp
1 #include <iostream>
2 using namespace std;
3 struct nodo
4 {
5     int num;
6     struct nodo* prox;
7 };
8
9 int main()
10 {
11     nodo *temp, *lista=NULL;
12
13     //primeiro nó
14     temp = new nodo;
15     temp->num = 23;
16     temp->prox=lista ;
17     lista= temp;
18
19     //segundo nó
20     temp = new nodo;
21     temp->num = 13;
22     temp->prox=lista ;
23     lista= temp;
24
25     //terceiro nó
26     temp = new nodo;
27     temp->num = 15;
28     temp->prox=lista ;
29     lista= temp;
30
31     //quarto nó
32     temp = new nodo;
33     temp->num = 18;
34     temp->prox=lista ;
35     lista= temp;
36
37     //listando
38     cout<<"\nInserindo No na Frente - Li tando um a um\n";
39     cout<<"\nValor do 1o no: "<<lista->num;
40     cout<<"\nValor do 2o no: "<<lista->prox->num;
41     cout<<"\nValor do 3o no: "<<lista->prox->prox->num;
42     cout<<"\nValor do 4o no: "<<lista->prox->prox->prox->num;
43
44     //liberando
45     delete lista;  lista=0;
46     cout<<"\n\n";
47     system("pause");
48 }

```

O momento decisivo!!!

A criação da função que insere na frente da Lista

Vamos fazer assim. Eu vou dando umas dicas e você vai tentando imaginar a função. Se preferir, vá escrevendo.

Ao final, clique no botão solução e compare sua resposta.

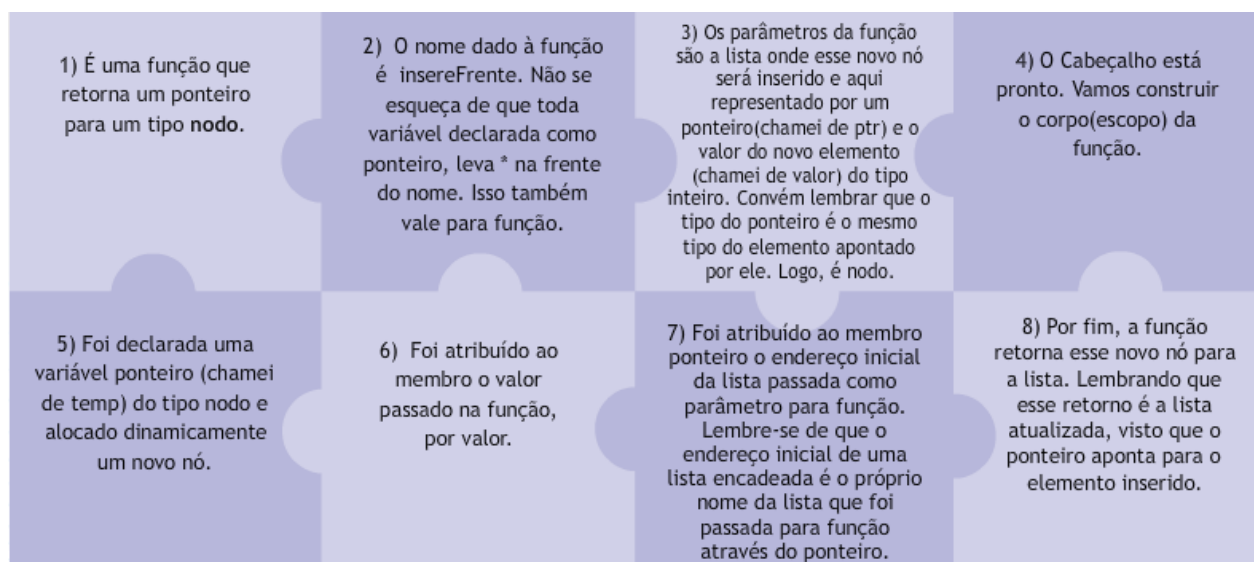
Estou torcendo por você!!!

DICAS

Admita a seguinte definição

```
struct nodo
{
    int num;
    struct nodo *prox;
};
```

Agora leia cada peça do quebra-cabeças para saber as dicas:



Já conseguiu escrever a função?! Agora compare a sua resposta no link a seguir:

http://estaciodocente.webaula.com.br/cursos/gon119/docs/08ED_DOC04.pdf

Cumpri minha parte fazendo o primeiro exercício. Agora é com você. Leia atentamente os dois exemplos e construa as funções. Depois, altere os códigos programas.

Esta é uma aula com um conteúdo muito mais complexo do que as anteriores porque o conceito de ponteiro, considerado pelos estudantes o “terror” da linguagem, tinha que ser passado a vocês de forma mais simples possível Não foi fácil, admito.

Leia com atenção o conteúdo da aula on-line e, se surgir dúvidas, fale com seu professor!

Até a próxima aula.

Saiba mais



Clique aqui para saber mais sobre o assunto:

<http://estaciodocente.webaula.com.br/cursos/gon119/docs/Arquivoscpp.pdf>

Cumpri minha parte fazendo o primeiro exercício. Agora é com você. Leia atentamente os dois exemplos e construa as funções. Depois altere os códigos dos programas.

Esta é uma aula com um conteúdo muito mais complexo do que as anteriores porque o conceito de ponteiro, considerado pelos estudantes o “terror” da linguagem, tinha que ser passado a vocês de forma mais simples possível. Não foi fácil, admito.

Leia com atenção o conteúdo da aula on-line e, se surgir dúvidas, fale com seu professor!

Até a próxima aula.

O que vem na próxima aula

- Na próxima aula você irá finalizar o estudo sobre Listas Encadeadas.

CONCLUSÃO

Nesta aula, você:

- Compreendeu a importância da Alocação Dinâmica;
- Compreendeu e usou a variável do tipo ponteiro;
- Analisou aplicações de Listas Lineares Encadeadas com poucos nós.