L03: Listas

Murilo Dantas

Exercícios

Grupo 1

1. **Faça um programa que cadastre 5 produtos. Para cada produto devem ser cadastrados código do produto, preço e quantidade estocada. Os dados devem ser armazenados em uma lista simplesmente encadeada e não ordenada. Posteriormente, receber do usuário a taxa de desconto (ex.: digitar 10 para taxa de desconto de 10%). Aplicar a taxa digitada ao preço de todos os produtos cadastrados e finalmente mostrar um relatório com o código e o novo preço. O final desse relatório deve apresentar também a quantidade de produtos com quantidade estocada superior a 500.**

Grupo 2

2. **Faça um programa que cadastre 8 funcionários. Para cada funcionário devem ser cadastrados nome e salário. Os dados devem ser armazenados em uma lista simplesmente encadeada e ordenada, de forma decrescente, pelo salário do funcionário. Posteriormente, o programa de mostrar:**
    a. **O nome do funcionário que tem o maior salário (em caso de empate mostrar todos);**
    b. **A média salarial de todos os funcionários juntos;**
    c. **A quantidade de funcionários com salário superior a um valor fornecido pelo usuário. Caso nenhum funcionário satisfaça essa condição, mostrar mensagem.**

Grupo 3

3. **Faça um programa que cadastre 5 alunos. Para cada aluno devem ser cadastrados nome e nota final. Os dados devem ser armazenados em uma lista duplamente encadeada e não ordenada. Em seguida, o programa deve mostrar apenas o nome dos alunos aprovados, ou seja, alunos com nota final de no mínimo 7. Se nenhum aluno estiver aprovado, mostrar mensagem.**

Grupo 4

4. **Faça um programa que cadastre o nome e o salário de 6 funcionários em uma lista duplamente encadeada e ordenada pelo salário de forma crescente. A seguir, o programa deve mostrar o nome, o valor do imposto e o valor a receber, ou seja, o salário menos o imposto de todos os funcionários cadastrados. Posteriormente, o programa deve mostrar os nomes e os salários dos funcionários cujos nomes comecem por uma letra digitada pelo usuário (considera a possibilidade de letras maiúsculas e minúsculas). Se nenhum funcionário tem o nome começado com a letra digitada pelo usuário, mostrar mensagem. Finalmente, o programa deve apresentar duas listagens:**
    a. **Dos nomes e salários dos funcionários por ordem crescente de salário;**

b. **Dos nomes e salários dos funcionários por ordem decrescente de salário.**

Observação: os percentuais de imposto seguem a tabela abaixo:

Valor do salário	Percentual do imposto
Até 850	Isento
Entre 850 e 1200	10% do salário
De 1200 para cima	20% do salário

Grupo 5

5. **Faça um programa que receba 20 números e armazene os pares em uma lista simplesmente encadeada e não ordenada e os ímpares em uma segunda lista simplesmente encadeada e não ordenada. Posteriormente, o programa deve montar uma terceira lista, duplamente encadeada e ordenada de forma crescente, com os números das duas listas anteriores. Para finalizar, o programa deve mostrar todos os números da terceira lista das seguintes formas:**
   a. **Crescente.**
   b. **Decrescente.**

Grupo 6

6. **Faça um programa que implemente uma lista encadeada de números inteiros com inserção de dados pelo usuário através de um menu. Escreva uma função que verifique se esta lista está em ordem crescente e outra função que faça uma busca na lista, dado um elemento passado pelo usuário. Faça versões recursiva e iterativa.**

Grupo 7

7. **Faça um programa que implemente uma lista encadeada de números inteiros com inserção de dados pelo usuário através de um menu. Escreva uma função que encontre o nó que contenha o menor valor e outra função que verifique se duas listas encadeadas são iguais, ou melhor, se têm o mesmo conteúdo. Faça duas versões: uma iterativa e uma recursiva.**

Grupo 8

8. **Faça um programa que implemente uma lista encadeada de números inteiros com inserção de dados pelo usuário através de um menu. Escreva uma função que faça uma cópia da lista e outra função que concatene duas listas encadeadas (isto é, engate a segunda no fim da primeira). Faça duas versões da função: uma iterativa e uma recursiva.**

Grupo 9

9. **Faça um programa que implemente uma lista encadeada de números inteiros com inserção de dados pelo usuário através de um menu. Escreva uma função que insira uma nova célula com conteúdo x imediatamente depois da k-ésima célula da lista encadeada e outra função que troque de posição duas células de uma mesma lista. Faça duas versões: uma iterativa e uma recursiva.**

Grupo 10

10. **Faça um programa que implemente uma lista encadeada de números inteiros com inserção de dados pelo usuário através de um menu. Escreva uma função que inverta a ordem das células de uma lista encadeada (a primeira passa a ser a última, a segunda passa a ser a penúltima etc.). Faça isso sem usar espaço auxiliar, apenas alterando ponteiros. Dê duas soluções: uma iterativa e uma recursiva.**

Grupo 10

11. **Faça um programa que implemente uma lista encadeada de números inteiros com inserção de dados pelo usuário através de um menu. Escreva uma função que copie o conteúdo de um vetor para uma lista encadeada preservando a ordem dos elementos e outra função que copie o conteúdo de uma lista encadeada para um vetor preservando a ordem dos elementos. Faça duas versões: uma iterativa e uma recursiva.**

Grupo 9

12. **Faça um programa que implemente uma lista duplamente encadeada de números inteiros com inserção de dados pelo usuário através de um menu. Escreva uma função que remova da lista a célula apontada por p e outra função que insira uma nova célula com conteúdo x nesta lista duplamente encadeada logo após a célula apontada por p. Dê duas soluções: uma iterativa e uma recursiva.**

Código exemplo

Nas notas de aula, foram apresentadas Estruturas de Dados usando **struct** em C. É possível também programar com Orientação a Objeto usando C++.

1. Crie um Projeto Vazio no Visual Studio (Visual C++).
2. Adicione os arquivos: No.h, No.cpp, Lista.h, Lista.cpp, Teste.h e Teste.cpp, e os preencha com os respectivos conteúdos, a seguir.

Declaração da classe No em No.h:

```
//Definição do nó para uma lista encadeada de inteiros
#include <iostream>
using namespace std;

class No
{
	int info;//a informação armazenada pelo nó é um inteiro
	No* proximo; //Ponteiro para o proximo nó

public:
	//Construtor
	No(int elem); //cria um nó com o valor dado por elem.

	//Métodos de Acesso
	int getInfo(); //lê o valor de info do nó
	void setInfo(int novoInfo); //modifica o valor do nó
```

```
        No* getProximo(); //retorna o proximo nó
        void setProximo(No *novoNo); //modifica o valor do proximo nó novoNo

        //Destrutor
        ~No(); //Destroi o nó.
};
```

A implementação da classe No é feita no arquivo No.cpp:

```
#include "No.h"

//Construtor
No::No(int elem)
{
        info = elem;
        proximo = NULL;
}

//Métodos de Acesso
int No::getInfo()
{
        return info;
}

void No::setInfo(int novoInfo)
{
        info = novoInfo;
}

No* No::getProximo()
{
        return proximo;
}

void No::setProximo(No *novoNo)
{
        proximo = novoNo;
}

        //Destrutor
No::~No()
{ //Se esta msg não aparecer no final, a memória alocada para o nó não foi liberada!
        cout << "Noh de valor " << info << " destruido!" << endl;
}
```

A seguir, a classe Lista é declarada em Lista.h:

```
#include "No.h"

class Lista
{ //Definição de uma lista de inteiros: ponteiro para o primeiro nó.
        No* primeiro;
```

```
public:
        //Construtor
        Lista(); //cria lista vazia

        //Métodos de Acesso
        No* getPrimeiro(); //retorna o primeiro nó da lista
        void setPrimeiro(No* novoNo); //torna "novoNo" o primeiro nó da lista

        //Métodos
        bool eVazia(); //retorna verdadeiro se a lista é uma lista vazia
        void insere(int elem); //insere elemento no início da lista
        No* remove(int elem); //remove todas as ocorrencias de elem na lista
        bool existe(int elem); //verifica se elem existe na lista
        void imprime(); //imprime todos os elementos da lista

        //Destrutor
        ~Lista(); //Destroi todos os nós da lista
} ;
```

A implementação de Lista é:

```
#include "Lista.h"

//Construtor
Lista::Lista()
{
        primeiro = NULL;
}

//Métodos de Acesso
No* Lista::getPrimeiro()
{
        return primeiro;
}

void Lista::setPrimeiro(No* novoNo)
{
        primeiro = novoNo;
}

        //Métodos
bool Lista::eVazia()
{
        if (primeiro == NULL)
                return true;
        return false;
}

void Lista::insere(int elem)
{
        No *segundo = primeiro; //armazena segundo nó da nova lista
        primeiro = new No(elem); //o primeiro elemento da lista passa a ser o novo nó;
        primeiro->setProximo(segundo);//o proximo nó da lista é aquele
// armazenado anteriormente
```

```
}

void Lista::imprime()
{
	if(eVazia())
		cout << "{ }";
	else
	{
		No* pAux=primeiro;

		cout << "{";
		while(true)
		{
			cout << pAux->getInfo();//impressão do nó atual
			if(pAux->getProximo() != NULL)//Se ainda existem nós...
			{
				cout << ", ";
				pAux = pAux->getProximo(); //passa-se ao próximo nó
			}
			else
			{ //fim da lista
				cout << "}";
				break;
			}
		}
	}
}

bool Lista::existe(int elem)
{
	if(eVazia())
		return false;

	No* pAux = primeiro;
	while(pAux != NULL)
	{
		if(pAux->getInfo() == elem)//elemento encontrado!
			return true;
		else
			pAux = pAux->getProximo(); //passa-se ao próximo nó
	}
	return false; //nenhum nó com valor "elem" foi encontrado
}

No* Lista::remove(int elem)
{
	if(eVazia())
		return NULL;//o metodo retorna um ponteiro para No...

	//Quando o elem é o primeiro da lista...
	while(primeiro->getInfo() == elem)
	{
		No* aux = primeiro;//armazena o primeiro a ser removido.
		primeiro = primeiro->getProximo(); //passa-se ao próximo nó,
  //que agora é o primeiro.
```

```
                aux->~No(); //destruição do nó removido
                delete aux; //destruição do nó removido de forma mais limpa
                if(primeiro == NULL) //se for encontrado fim de lista....
                        return primeiro;
        }

        //O primeiro não é mais um elemento a ser removido...
        Lista *pAux = new Lista(); //Cria lista auxiliar...

        pAux->primeiro = primeiro->getProximo();//... que se inicia no
//proximo elemento da lista atual

        primeiro->setProximo(pAux->remove(elem)); //remove elem da  nova lista,
// associando seu início à lista atual

        return primeiro;
}

//Destrutor
Lista::~Lista()
{//A lista deve ser destruída nó por nó.
        No* pAux;
        while(primeiro != NULL)
        {
                pAux = primeiro;
                primeiro = primeiro->getProximo();
                pAux->~No();
        }
}
```

O teste está dividido em dois arquivos:
Teste.h:

```
#include "Lista.h"

const int SAIR = 0;
const int IMPRIMIR = 1;
const int INSERIR = 2;
const int APAGAR = 3;
const int PROCURAR = 4;

int menu(); //função que apresenta o menu para o
//usuário e retorna a escolha deste
```

Teste.cpp:

```
#include "Teste.h"

void main(){

        cout << "++++++++++++++++++++++++++++\n";
        cout << "Programa de Teste de Listas\n";
        cout << "++++++++++++++++++++++++++++\n";
```

```
	Lista lst;
	int n, opcao;

	while((opcao=menu()) != SAIR)
	{
		switch(opcao)
		{
			case IMPRIMIR:
				lst.imprime();
				cout << endl;
				break;
			case INSERIR:
				cout << "n? ";
				cin >> n;
				lst.insere(n);
				break;
			case APAGAR:
				cout << "n? ";
				cin >> n;
				lst.remove(n);
				break;
			case PROCURAR:
				cout << "n? ";
				cin >> n;
				if(lst.existe(n))
					cout << "O elemento estah na lista!\n";
				else
					cout << "Este elemento nao estah na lista...\n";
				break;
			case SAIR:
				break;
			default:
				cout << " Opcao Invalida! \n";
		} //fim do switch
	} //fim do while


} //fim do main

int menu()
{
	int opcao;

	cout << IMPRIMIR << " - imprimir a lista\n";
	cout << INSERIR << " - inserir um elemento na lista\n";
	cout << APAGAR << " - apagar um elemento da lista\n";
	cout << PROCURAR << " - procurar um elemento na lista\n";
	cout << SAIR << " - encerrar o programa\n";

	cout << "Opcao ? ";
	cin >> opcao;

	return opcao;
}
```

Grupo 8

13. **Implemente um novo método para a classe Lista chamado “total” que retorne o número de elementos na lista.**

Grupo 7

14. **Implemente um novo método para a classe Lista chamado também “total” que retorne o número de vezes que um elemento dado como parâmetro aparece na lista (se aparecer).**

Grupo 6

15. **Implemente um novo método para a classe Lista chamado “posicao” que retorne a posição em que um elemento aparece pela primeira vez em uma lista. Se o elemento não aparece, o método deve retornar zero.**

Grupo 5

16. **Implemente o método “copia” que faz uma cópia de uma lista nó por nó, retornando a nova lista. A cópia deve ser independente do original, ou seja, alterações em uma não devem refletir na outra. Atenção: Cuidado com a ordem em que os elementos são inseridos! DICA: crie um método insereNoFinal para fazer a inserção de elementos no final da lista, e não no início.**

Grupo 4

17. **Crie um método remove não recursivo, que retorne void. (DICA: Crie um segundo ponteiro auxiliar para apontar para o elemento anterior ao que está sendo percorrido).**

Grupo 3

18. **Crie uma lista ordenada de inteiros sobrepondo os métodos insere e remove de Lista. Para testar, faça um programa que gere aleatoriamente um número qualquer (dado pelo usuário) de elementos, imprima a ordem em que eles foram gerados e a ordem em que eles aparecem na lista.**

Grupo 2

19. **Crie uma lista duplamente encadeada de inteiros que possa ser percorrida na ordem normal e na ordem inversa. Faça um programa de teste apropriado.**

Grupo 1

20. **Crie uma lista circular de inteiros derivando a Lista do exemplo. Para testar, gere uma sequência de números aleatórios como no exercício anterior e imprima a lista a partir de um ponto selecionado pelo usuário.**