Alunos: Amós Cruz Dos Santos e Emerson Santos Coutinho

Questões:1 2 3 4 5 6 7 8 9 14 15 16 23 25

1:

Aqui adotamos uma regra na função de inserção:

Se x == 2 adicionamos o valor i no fim da lista. Se x == 1, adicionamos no início.

Nota:Adotamos que: x = 1 adicionamos o valor no começo da lista; x = 2 adicionamos no final;

#include <iostream>

using namespace std;

struct No{

int i;

No \* p, \* a;

}\*ini = NULL, \*fim = NULL;

void insert(int i, int x);

int imprime();

int exist(int i);

int main()

{

int num = 0;

insert(10, 2);

insert(20, 2);

insert(30, 2);

insert(40, 2);

insert(50, 2);

imprime();

insert(14, 2);

cout << "\nNumero:";

cin >> num;

insert(num, 1);

imprime();

cout << "valor 14 aparece na posição:" << exist(14);

return 0;

}

void insert(int i, int x)

{

No \* novono = (No \*)malloc(sizeof(No \*));

novono->i = i;

if(ini == NULL and fim== NULL)

{

novono->a = NULL;

novono->p = NULL;

ini = novono;

fim = novono;

}

else if(ini->a == fim->a and ini->p == fim->p)

{

No \* aux = ini;

if(x == 1)

{

novono->a = NULL;

novono->p = aux;

aux->a = novono;

ini = novono;

fim = aux;

}

else

{

novono->a = aux;

novono->p = NULL;

aux->p = novono;

fim = novono;

ini = aux;

}

}

else

{

No \* aux = ini;

if(x==2)

{

while(aux->p!= NULL){

aux = aux->p;

}

}

if(aux->p == NULL)

{

aux->p = novono;

novono->a = aux;

novono->p = NULL;

fim = novono;

}

else if(aux->a == NULL)

{

aux->a = novono;

novono->p = aux;

novono->a = NULL;

ini = novono;

}

}

}

int imprime()

{

No \* aux = ini;

do

{

cout << aux->i << " ";

aux = aux->p;

}while(aux != NULL);

}

int exist(int i)

{

No \* aux = ini;

int cont\_pos = 0;

while(aux != NULL)

{

if(aux->i == i)

{

return cont\_pos;

}

aux = aux->p;

cont\_pos++;

}

cout << endl <<"Dont Ocorrs" << endl;

return 0;

}

Letra F:

A ordem dos elementos na lista não muda em nada a implementação dela nesse exemplo. Se os valores forem inseridos em ordem crescente ou decrescente não afetaria a inserção de outros valores no começo ou no fim dela. Só haveria mudança se a inserção destes exigisse uma ordem.

2:

Aqui adotamos a seguinte ideia:

Mudamos a função exist do exercício anterior para verificar a existência sem considerar a posição. Usamos ela como limitador na função insert que só adiciona quando exist retornar 0.

Nota:Adotamos que: x = 1 adicionamos o valor no começo da lista; x = 2 adicionamos no final;

void insert(int i, int x)

{

if(exist(i) == 0)

{

No \* novono = (No \*)malloc(sizeof(No \*));

novono->i = i;

if(ini == NULL and fim== NULL)

{

novono->a = NULL;

novono->p = NULL;

ini = novono;

fim = novono;

}

else if(ini->a == fim->a and ini->p == fim->p)

{

No \* aux = ini;

if(x == 1)

{

novono->a = NULL;

novono->p = aux;

aux->a = novono;

ini = novono;

fim = aux;

}

else

{

novono->a = aux;

novono->p = NULL;

aux->p = novono;

fim = novono;

ini = aux;

}

}

else

{

No \* aux = ini;

if(x==2)

{

while(aux->p!= NULL){

aux = aux->p;

}

}

if(aux->p == NULL)

{

aux->p = novono;

novono->a = aux;

novono->p = NULL;

fim = novono;

}

else if(aux->a == NULL)

{

aux->a = novono;

novono->p = aux;

novono->a = NULL;

ini = novono;

}

}

}

}

int exist(int i)

{

No \* aux = ini;

while(aux != NULL)

{

if(aux->i == i)

{

return 1;

}

aux = aux->p;

}

return 0;

}

3:

Aqui nós modificamos o método Insert para que ele adicione o elemento na posição adequada na lista. Dividimos em 3 Ifs para considerar: Lista vazia, lista com um elemento e lista com apenas 1 elemento e lista com mais de 2 elementos.

Nota: O código e as função são as mesmas usadas na questão 1, a mudança ocorre apenas no método insert que passa a adicionar a partir de uma ordem.

void insert(int i)

{

No \* novono = (No \*)malloc(sizeof(No \*));

novono->i = i;

if(ini == NULL and fim== NULL)

{

novono->a = NULL;

novono->p = NULL;

ini = novono;

fim = novono;

}

else if(ini->a == fim->a and ini->p == fim->p)

{

No \* aux = ini;

if(aux->i > i)

{

novono->a = NULL;

novono->p = aux;

aux->a = novono;

ini = novono;

fim = aux;

}

else

{

novono->a = aux;

novono->p = NULL;

aux->p = novono;

fim = novono;

ini = aux;

}

}

else

{

No \* aux = ini, \* aux2;

while(aux->p!= NULL){

if(aux->i > i)

break;

aux2 = aux;

aux = aux->p;

}

if(aux->i < i and aux->p == NULL)

{

aux->p = novono;

novono->a = aux;

novono->p = NULL;

fim = novono;

}

else if(aux->i > i and aux->a == NULL)

{

aux->a = novono;

novono->p = aux;

novono->a = NULL;

ini = novono;

}else

{

aux2->p = novono;

novono->a = aux2;

aux->a = novono;

novono->p = aux;

}

}

}

4:

Poderíamos representar com uma struct:

struct No{

int matricula, idade;

char \* nome;

No \* p, \* a;

};

5:

No metodo insert nos verificamos se o aluno existe, com a função exist, antes de adicionar na lista.

Nota: Adotamos que: x = 1 adicionamos o valor no começo da lista; x = 2 adicionamos no final;

#include <string.h>

#include <iostream>

using namespace std;

struct No{

int matricula, idade;

char \* nome;

No \* p, \* a;

}\*ini = NULL, \*fim = NULL;

void insert(int matricula, int idade, char nome[25],int x);

int imprime();

int exist(int i);

int main()

{

int num = 0, matricula, idade;

char nome[25];

for(int i = 0;i<40;i++)

{

cout << "Matricula" << endl;

cin >> matricula;

cout << "Idade:" << endl;

cin >> idade;

cout << "Nome:" << endl;

cin >> nome;

insert(matricula, idade, nome, 2);

}

cout << "\nAlunos cadastrados:" << endl;

imprime();

return 0;

}

void insert(int matricula, int idade, char nome[25],int x)

{

if(exist(matricula) == 0)

{

No \* novono = (No \*)malloc(sizeof(No \*));

novono->matricula = matricula;

novono->idade = idade;

novono->nome = (char \*)malloc(sizeof(char) \* 25);

strcpy(novono->nome, nome);

if(ini == NULL and fim== NULL)

{

novono->a = NULL;

novono->p = NULL;

ini = novono;

fim = novono;

}

else if(ini->a == fim->a and ini->p == fim->p)

{

No \* aux = ini;

if(x == 1)

{

novono->a = NULL;

novono->p = aux;

aux->a = novono;

ini = novono;

fim = aux;

}

else

{

novono->a = aux;

novono->p = NULL;

aux->p = novono;

fim = novono;

ini = aux;

}

}

else

{

No \* aux = ini;

if(x==2)

{

while(aux->p!= NULL){

aux = aux->p;

}

}

if(aux->p == NULL)

{

aux->p = novono;

novono->a = aux;

novono->p = NULL;

fim = novono;

}

else if(aux->a == NULL)

{

aux->a = novono;

novono->p = aux;

novono->a = NULL;

ini = novono;

}

}

}else

{

cout << "Matricula já consta na lista, aluno removido." << endl;

cout << "Informacoes do aluno:" << endl;

cout << "Nome:" << nome << endl;

cout << "Matricula:" << matricula << endl;

cout << "Idade:" << idade << endl;

}

}

int imprime()

{

if(ini != NULL)

{

No \* aux = ini;

do

{

cout <<"Matricula:"<< aux->matricula<< " ";

cout <<"Nome:"<< aux->nome<< " ";

cout <<"idade:"<< aux->idade<< endl;

aux = aux->p;

}while(aux != NULL);

}

}

int exist(int matricula)

{

if(ini != NULL)

{

No \* aux = ini;

while(aux != NULL)

{

if(aux->matricula == matricula)

{

return 1;

}

aux = aux->p;

}

return 0;

}

return 0;

}

6:

Aqui nós usamos uma função auxiliar “remove” que vai remover um elemento i da lista. Assim, dentro da função retira, buscamos pelos alunos com idade menor que N e passamos para “remove” tirá-lo da lista.

Nota: O código e as função são as mesmas usadas na questão 1, adicionando apenas o método remove e o retira.

void remove(int i)

{

if(exist(i))

{

No \* aux = ini, \* aux2;

while(aux->p!= NULL){

if(aux->matricula == i)

break;

aux2 = aux;

aux = aux->p;

}

if(aux->p == NULL)

{

aux2->p = NULL;

fim = aux2;

free(aux);

}

else if(aux->a == NULL)

{

aux2 = aux->p;

aux2->a = NULL;

ini = aux2;

free(aux);

}else

{

aux2->p = aux->p;

aux2 = aux->p;

aux2->a = aux->a;

free(aux);

}

}

}

void retira(int i)

{

if(ini != NULL)

{

No \* aux = ini;

do

{

if(aux->idade < i)

remove(aux->matricula);

aux = aux->p;

}while(aux != NULL);

}

}

int exist(int matricula)

{

if(ini != NULL)

{

No \* aux = ini;

while(aux != NULL)

{

if(aux->matricula == matricula)

{

return 1;

}

aux = aux->p;

}

return 0;

}

return 0;

}

7.

Nesse exercício, declaramos um vetor de struct list para ser a nossa segunda lista e recebemos a quantidade de elementos dentro de concat. À medida que passamos pela lista nós chamamos a função insert que vai adicionar na lista os valores do vetor.

Nota: O código e as funções são as mesmas usadas na questão 1, adicionando apenas a declaração do vetor na struct e o método concat.

struct No{

int i;

No \* p, \* a;

}\*ini = NULL, \*fim = NULL, list[3];

void concact(int l)

{

for(int i = 0;i<l;i++)

{

insert(list[i].i, 2);

}

}

8:

Primeiro verificamos com os Ifs se: a lista está vazia ou a lista está com um só nó. Depois usamos um while para passar pelos elementos e encontrar a posição do maior.

Nota: O código e as funções são as mesmas usadas na questão 1, adicionando apenas o método transfereMaior.

void transfereMaior()

{

if(ini != NULL)

{

if(ini->p != NULL)

{

No \* aux = ini, \* maior = ini, \*ma, \*mp;

//ma = membro anterior e mp = membro posterior da lista

while(aux!=NULL)

{

if(maior->valor < aux->valor)

{

ma = aux->a;

mp = aux->p;

maior = aux;

}

aux = aux->p;

}

if(ini->valor != maior->valor)

{

//If abaixo evita que usemos um endereço invalido caso estivermos na ultima posição da lista

if(mp!= NULL)

mp->a = maior->a;

ma->p = maior->p;

ini->a = maior;

maior->p = ini;

maior->a = NULL;

ini = maior;

}

else

{

cout << "Maior valor já está no inicio." << endl;

}

}else

{

cout << "Lista contem apenas 1 no" << endl;

}

}else

{

cout << "Lista vazia" << endl;

}

}

9:

Na adição de valores, usamos 0 para adicionar no inicio, -1 para o fim e um valor x para adicionar em uma posição da lista. Exemplo:

Para a lista:

1 2 3 5

se rodarmos insert(10, -1) teremos:

1 2 3 5 10

se rodarmos insert(4, 3) teremos:

1 2 3 4 5 10

se rodarmos insert(0, 0) teremos:

0 1 2 3 4 5 10

#include <string.h>

#include <iostream>

using namespace std;

struct No{

int valor;

No \* p, \* a;

}\*ini = NULL, \*fim = NULL;

void insert(int v, int x);

int imprime();

int main()

{

int valor = 0, x = 0;

insert(5, -1);

insert(8, -1);

insert(20, -1);

insert(30, -1);

cout << "Lista 01:" << endl;

for(int i = 0;i<5;i++)

{

cout << "Digite um valor:" << endl;

cin >> valor;

cout << "x:";

cin >> x;

insert(valor, x);

cout << "Lista até o momento:" << endl;

imprime();

}

return 0;

}

void insert(int valor,int x)

{

No \* novono = (No \*)malloc(sizeof(No \*));

novono->valor = valor;

if(ini == NULL and fim== NULL)

{

novono->a = NULL;

novono->p = NULL;

ini = novono;

fim = novono;

}

else if(ini->a == fim->a and ini->p == fim->p)

{

if(x==0)

{

No \* aux = ini;

novono->a = NULL;

novono->p = aux;

aux->a = novono;

fim = aux;

ini = novono;

}

else

{

No \* aux = ini;

novono->a = aux;

novono->p = NULL;

aux->p = novono;

fim = novono;

ini = aux;

}

}

else

{

No \* aux = ini, \*aux2 = NULL;

if(x == 0){

aux->a = novono;

novono->p = aux;

novono->a = NULL;

ini = novono;

}

else if(x == -1)

{

while(aux->p!= NULL){

aux = aux->p;

}

aux->p = novono;

novono->a = aux;

novono->p = NULL;

fim = novono;

}

else

{

for(int i = 0;i!=x and aux != NULL;i++)

{

aux2 = aux;

aux = aux->p;

}

if(aux != NULL and x > -2)

{

aux2->p = novono;

novono->a = aux2;

aux->a = novono;

novono->p = aux;

}else

{

cout << "Posicao fora do length." << endl;

}

}

}

}

int imprime()

{

if(ini != NULL)

{

No \* aux = ini;

do

{

cout << aux->valor<< " ";

aux = aux->p;

}while(aux != NULL);

}

cout << endl;

}

14:

Pilha feita em sala, fizemos apenas a adição do código na função main.

#include <iostream>

using namespace std;

struct No{

int i = 0;

No \* next;

}\* topo = NULL;

void push(int i);

int isEmpty();

void pop();

int top();

int main()

{

int valor = 0;

for(int i = 0;i<5;i++)

{

cout << "Digite o " << i+1 << " valor:";

cin >> valor;

if(valor % 2 == 0)

{

push(valor);

}

else

{

pop();

}

}

cout << "Pilha:" << endl;

while(!isEmpty())

{

cout << top() << endl;

pop();

}

return 0;

}

void push(int i)

{

No \* novono = (No \*)malloc(sizeof(No \*));

novono->i = i;

if(topo ==NULL)

{

novono->next = NULL;

topo = novono;

}

else

{

novono->next = topo;

topo = novono;

}

}

int isEmpty()

{

if(topo == NULL)

return 1;

else

return 0;

}

void pop()

{

if(topo != NULL)

{

No \* aux;

aux = topo;

topo = topo->next;

free(aux);

}

}

int top()

{

if(topo!= NULL)

return topo->i;

}

15:

A maneira mais simples de resolver problemas com duas pilhas é criando duas pilhas e usando IFs que executem a partir de um valor x. Usamos a seguinte regra: pinha P= topo e pilha N = topo2;

#include <iostream>

using namespace std;

struct No{

int i = 0;

No \* next;

}\* topo = NULL, \*topo2 = NULL;

void push(int i, int x);

int isEmpty(int x);

void pop(int x);

int top(int x);

int main()

{

int valor = 0;

for(int i = 0;i<5;i++)

{

cout << "Digite o " << i+1 << " valor:";

cin >> valor;

if(valor > 0)

{

push(valor, 1);

}

else if(valor < 0)

{

push(valor, 2);

}else

{

pop(1);

pop(2);

}

}

cout << "Pilha 01:";

while(!isEmpty(1))

{

cout << top(1) << " ";

pop(1);

}

cout << endl << "Pilha 02:";

while(!isEmpty(2))

{

cout << top(2) << " ";

pop(2);

}

cout << endl;

return 0;

}

void push(int i, int x)

{

if(x == 1)

{

No \* novono = (No \*)malloc(sizeof(No \*));

novono->i = i;

if(topo ==NULL)

{

novono->next = NULL;

topo = novono;

}

else

{

novono->next = topo;

topo = novono;

}

}

else

{

No \* novono = (No \*)malloc(sizeof(No \*));

novono->i = i;

if(topo2 ==NULL)

{

novono->next = NULL;

topo2 = novono;

}

else

{

novono->next = topo2;

topo2 = novono;

}

}

}

int isEmpty(int x)

{

if(x == 1)

{

if(topo == NULL)

return 1;

else

return 0;

}

else

{

if(topo2 == NULL)

return 1;

else

return 0;

}

}

void pop(int x)

{

if(x == 1)

{

if(topo != NULL)

{

No \* aux;

aux = topo;

topo = topo->next;

free(aux);

}

}

else

{

if(topo2 != NULL)

{

No \* aux;

aux = topo2;

topo2 = topo2->next;

free(aux);

}

}

}

int top(int x)

{

if(x == 1)

{

if(topo != NULL)

return topo->i;

}

else

{

if(topo2 != NULL)

return topo2->i;

}

}

16:

Aqui adotamos a seguinte ideia:

Como as pilhas estão ordenadas a partir do topo, ou seja:

Pilha 01:

0

1

2

3

4

Pilha 02:

5

6

7

8

9

Assim, basta para que a pilha 3 fique ordenada em ordem decrescente desempilhamos a pilha 01:

pilha 03: 4 3 2 1 0

E depois a pilha 02:

pilha 03:

9

8

7

6

5

4

3

2

1

0

#include <iostream>

using namespace std;

struct No{

int i = 0;

No \* next;

}\* topo = NULL, \*topo2 = NULL, \*topo3=NULL;

void push(int i, int x);

int isEmpty(int x);

void pop(int x);

int top(int x);

int main()

{

for(int i = 4;i>-1;i--)

{

push(i, 1);

}

for(int i = 9;i>4;i--)

{

push(i, 2);

}

cout << "Pilha 01:";

while(!isEmpty(1))

{

cout << top(1) << " ";

push(top(1), 3);

pop(1);

}

cout << endl << "Pilha 02:";

while(!isEmpty(2))

{

cout << top(2) << " ";

push(top(2), 3);

pop(2);

}

cout << endl;

cout << "Pilha 03:";

while(!isEmpty(3))

{

cout << top(3) << " ";

pop(3);

}

cout << endl;

return 0;

}

void push(int i, int x)

{

if(x == 1)

{

No \* novono = (No \*)malloc(sizeof(No \*));

novono->i = i;

if(topo ==NULL)

{

novono->next = NULL;

topo = novono;

}

else

{

novono->next = topo;

topo = novono;

}

}

else if(x == 2)

{

No \* novono = (No \*)malloc(sizeof(No \*));

novono->i = i;

if(topo2 ==NULL)

{

novono->next = NULL;

topo2 = novono;

}

else

{

novono->next = topo2;

topo2 = novono;

}

}

else

{

No \* novono = (No \*)malloc(sizeof(No \*));

novono->i = i;

if(topo3==NULL)

{

novono->next = NULL;

topo3= novono;

}

else

{

novono->next = topo3;

topo3= novono;

}

}

}

int isEmpty(int x)

{

if(x == 1)

{

if(topo == NULL)

return 1;

else

return 0;

}

else if(x==2)

{

if(topo2 == NULL)

return 1;

else

return 0;

}

else

{

if(topo3 == NULL)

return 1;

else

return 0;

}

}

void pop(int x)

{

if(x == 1)

{

if(topo != NULL)

{

No \* aux;

aux = topo;

topo = topo->next;

free(aux);

}

}

else if(x==2)

{

if(topo2 != NULL)

{

No \* aux;

aux = topo2;

topo2 = topo2->next;

free(aux);

}

}

else

{

if(topo3 != NULL)

{

No \* aux;

aux = topo3;

topo3 = topo3->next;

free(aux);

}

}

}

int top(int x)

{

if(x == 1)

{

if(topo != NULL)

return topo->i;

}

else if(x==2)

{

if(topo2 != NULL)

return topo2->i;

}else

{

if(topo3 != NULL)

return topo3->i;

}

}

23:

Desempenhamos os valores da pilha s1 e empilhamos na pilha auxiliar. Depois desempenhamos da auxiliar e empilhamos na s2;

//topo = s1, s2 = topo3, aux = topo2

#include <iostream>

using namespace std;

struct No{

int i = 0;

No \* next;

}\* topo = NULL, \*topo2 = NULL, \*topo3 = NULL;

void push(int i, int x);

int isEmpty(int x);

void pop(int x);

int top(int x);

int main()

{

for(int i = 0;i<5;i++)

{

push(i, 1);

}

cout << "PIlha s1:";

while(!isEmpty(1))

{

cout << top(1) << " ";

push(top(1), 2);

pop(1);

}

cout << endl;

cout << "PIlha Auxiliar:";

while(!isEmpty(2))

{

cout << top(2) << " ";

push(top(2), 3);

pop(2);

}

cout << endl;

cout << "PIlha s2:";

while(!isEmpty(3))

{

cout << top(3) << " ";

pop(3);

}

cout << endl;

return 0;

}

void push(int i, int x)

{

if(x == 1)

{

No \* novono = (No \*)malloc(sizeof(No \*));

novono->i = i;

if(topo ==NULL)

{

novono->next = NULL;

topo = novono;

}

else

{

novono->next = topo;

topo = novono;

}

}

else if(x == 2)

{

No \* novono = (No \*)malloc(sizeof(No \*));

novono->i = i;

if(topo2 ==NULL)

{

novono->next = NULL;

topo2 = novono;

}

else

{

novono->next = topo2;

topo2 = novono;

}

}

else

{

No \* novono = (No \*)malloc(sizeof(No \*));

novono->i = i;

if(topo3==NULL)

{

novono->next = NULL;

topo3= novono;

}

else

{

novono->next = topo3;

topo3= novono;

}

}

}

int isEmpty(int x)

{

if(x == 1)

{

if(topo == NULL)

return 1;

else

return 0;

}

else if(x==2)

{

if(topo2 == NULL)

return 1;

else

return 0;

}

else

{

if(topo3 == NULL)

return 1;

else

return 0;

}

}

void pop(int x)

{

if(x == 1)

{

if(topo != NULL)

{

No \* aux;

aux = topo;

topo = topo->next;

free(aux);

}

}

else if(x==2)

{

if(topo2 != NULL)

{

No \* aux;

aux = topo2;

topo2 = topo2->next;

free(aux);

}

}

else

{

if(topo3 != NULL)

{

No \* aux;

aux = topo3;

topo3 = topo3->next;

free(aux);

}

}

}

int top(int x)

{

if(x == 1)

{

if(topo != NULL)

return topo->i;

}

else if(x==2)

{

if(topo2 != NULL)

return topo2->i;

}else

{

if(topo3 != NULL)

return topo3->i;

}

}

25:

Aqui nós tiramos os valores da fila e empilhamos na pilha. Depois desempilhamos e adicionamos na fila.

#include <iostream>

using namespace std;

struct No{

int i;

No \* next;

}\* topo = NULL;

struct NoF{

int i;

NoF \* prox;

}\* ultimo = NULL;

void push(int i);

int isEmpty();

void pop();

int top();

int empty();

void adicionar(int i);

int imprime();

int remover();

int main()

{

int v = 0;

for(int i = 0;i<5;i++)

{

adicionar(i);

}

cout << "Fila:";

imprime();

while(!empty())

{

v = remover();

push(v);

}

cout << "\nPilha:";

while(!isEmpty())

{

cout << top() << " ";

adicionar(top());

pop();

}

cout << endl;

cout << "Fila final:";

imprime();

cout << endl;

return 0;

}

void adicionar(int i)

{

NoF \* novono =(NoF \*)malloc(sizeof(NoF \*));

novono->i = i;

if(ultimo == NULL)

{

novono->prox = NULL;

ultimo = novono;

}

else

{

novono->prox = ultimo;

ultimo = novono;

}

}

int empty()

{

if(ultimo == NULL)

return 1;

else

return 0;

}

int imprime()

{

NoF \* aux = ultimo;

while(aux!= NULL)

{

cout << aux->i << " ";

aux = aux->prox;

}

}

int remover()

{

int v = 0;

NoF \* aux = ultimo, \* aux2;

if(!empty())

{

if(aux->prox == NULL)

{

v = aux->i;

free(aux);

ultimo = NULL;

return v;

}

else

{

while(aux->prox != NULL)

{

aux2 = aux;

aux = aux->prox;

}

v = aux->i;

free(aux);

aux2->prox = NULL;

return v;

}

}

}

void push(int i)

{

No \* novono = (No \*)malloc(sizeof(No \*));

novono->i = i;

if(topo ==NULL)

{

novono->next = NULL;

topo = novono;

}

else

{

novono->next = topo;

topo = novono;

}

}

int isEmpty()

{

if(topo == NULL)

return 1;

else

return 0;

}

void pop()

{

if(topo != NULL)

{

No \* aux;

aux = topo;

topo = topo->next;

free(aux);

}

}

int top()

{

if(topo != NULL)

return topo->i;

}