6-4-2022

Practica 5 Lista, pila y cola simplemente dinámica con encabezado

Materia: Seminario de estructura de datos 1

Sección: D13.

Código: 216584703

Carrera: Ingeniería en computación.

Nombre alumno: Padilla Pérez Jorge Daray

Nombre profesor: Julio Esteban Valdes Lopez



**Introducción**

En esta practica se realizó la implementación de una Lista simplemente ligada dinámica en la que se incluyen las funciones de una pila y una cola, además de que incluye un encabezado que tiene el apuntador a siguiente, uno a anterior, una al primer nodo, otro al ultimo y un contador de los nodos existentes.

El programa está hecho con clases protegidas utilizando herencia y polimorfismo para usar correctamente la clase lista, en la cual se incluye los métodos básicos de una lista, además de tener los métodos de la pila y cola anteriormente mencionadas.

Texto

Descripción generada automáticamente

Aquí se aprecia el menú.

Texto

Descripción generada automáticamente

Si queremos insertar en una posición invalida no nos deja, ya que tiene que ser lineal y en este caso debería empezar en la posición 0.

Texto

Descripción generada automáticamente

Aquí insertamos un nodo en la posición 0.

Texto

Descripción generada automáticamente

Aquí insertamos otro nodo en la posición 0, ósea al principio de la lista.

Texto

Descripción generada automáticamente

Aquí mostramos la lista.

Texto

Descripción generada automáticamente

Insertamos un tercer nodo en la ultima posición.

Texto

Descripción generada automáticamente

Insertamos un nodo 4 en la posición 1 por lo que recorre a los demás.

Texto

Descripción generada automáticamente

Aquí usamos la función recupera que en este caso pide el nombre del nodo.

Texto

Descripción generada automáticamente

Aquí usamos la función localiza que en este caso pide la posicion del nodo.

Texto

Descripción generada automáticamente

Aquí eliminamos el pin 2.

Texto

Descripción generada automáticamente

Aquí anulamos la lista.

**Conclusión**

Se pudo completar de manera correcta el programa, utilizando una estructura dirección, en la que se almacenan los datos de la persona, seguido de una herencia a una clase llamada lista simple, en la cual pues se tienen los métodos principales de una lista simple como la que se pidió, además de tener los métodos de las pilas y las colas.

También se desarrollo con ayuda de el apuntador a anterior hacer una lista invertida, que me puede servir por si se pide implementar una lista circular o algo por el estilo, además también se implementó la eliminación del ultimo nodo por si se necesita en otro programa.

**Codigo fuente**

#include <iostream>

#include <string.h>

**using** **namespace** std;

**class** **StructBase**

{

protected:

**typedef** **struct** Address

{

char name[50];

char street[100];

char city[50];

char state[20];

int pin;

**struct** Address \*next;

**struct** Address \*before;

}Address;

Address \* first;

Address \* last;

int contador;

};

**class** **List**: StructBase

{

protected:

void inicializa()

{

first = nullptr;

last = nullptr;

contador = 0;

}

void show()

{

Address \*temp = (Address \*) malloc(**sizeof**(Address));

temp = first;

**if** (first != NULL){

cout << "Tamaño :"<<contador<<endl;

int i = 0;

**while** (temp != NULL){

printf("Actual %d", i);

printf("**\n** Name: %s**\n**", temp->name);

printf("street: %s**\n**", temp->street);

printf("City: %s**\n**", temp->city);

printf("State: %s**\n**", temp->state);

printf("pin: %i**\n\n**", temp->pin);

temp = temp ->next;

i++;

}

}

**else**{

printf("**\n** The list is empty**\n**");

}

}

void seekNode()

{

Address \*temp = (Address \*) malloc(**sizeof**(Address));

temp = first;

int found = 0 ;

char cadena[50];

printf(" Introduce the name of the node that you want to seek: ");

scanf("%s", &cadena);

**if** (first != NULL){

int i=0;

**while** (temp != NULL && found != 1){

**if** ( strcmp(temp->name, cadena)==0 ){

printf("Actual %d", i);

printf("**\n** Name: %s**\n**", temp->name);

printf("street: %s**\n**", temp->street);

printf("City: %s**\n**", temp->city);

printf("State: %s**\n**", temp->state);

printf("pin: %i**\n\n**", temp->pin);

found = 1;

}

temp = temp ->next;

i++;

}

**if** (found == 0){

printf("El nodo no fue encontrado");

}

}

**else**{

printf("**\n** The list is empty**\n**");

}

}

void Localiza()

{

Address \*temp = (Address \*) malloc(**sizeof**(Address));

temp = first;

int found = 0 ;

int pos;

printf(" Introduce the pos of the node that you want to locate: ");

scanf("%d", &pos);

**if** (first != NULL){

int i=0;

**while** (temp != NULL && found != 1){

**if** ( i == pos ){

printf("Actual %d", i++);

printf("**\n** Name: %s**\n**", temp->name);

printf("street: %s**\n**", temp->street);

printf("City: %s**\n**", temp->city);

printf("State: %s**\n**", temp->state);

printf("pin: %i**\n\n**", temp->pin);

found = 1;

}

temp = temp ->next;

i++;

}

**if** (found == 0){

printf("El nodo no fue encontrado");

}

}

**else**{

printf("**\n** The list is empty**\n**");

}

}

int tam()

{

**return** contador;

}

void eliminateNode()

{

Address \*actual = (Address \*) malloc(**sizeof**(Address));

actual = first;

Address\* before = (Address \*) malloc(**sizeof**(Address));

before = NULL;

int soughtnode = 0, found = 0;

printf(" Introduce the pin of the node that you want to eliminate: ");

scanf("%d", &soughtnode);

**if**(first != NULL){

**while**(actual != NULL && found != 1){

**if**(actual -> pin == soughtnode){

**if**(actual == first){

first = first ->next;

}

**else** **if** (actual == last)

{

last = before;

before -> next = actual -> next;

}

**else**{

before -> next = actual -> next;

}

printf("**\n**The link of the node have been eliminated");

found = 1;

}

before = actual;

actual = actual ->next;

}

**if**(found == 0){

printf("**\n**the node was not found**\n\n**");

}

**else**{

free(before);

contador--;

printf("**\n\n**NODE ELIMINATED SUCCESSFUL");

}

}

**else**{

printf("**\n**THE LIST IS EMPTY**\n\n**");

}

printf("**\n**");

}

void inserfirst()

{

Address \*in\_first = (Address \*) malloc(**sizeof**(Address));

**if**(!in\_first){*// new\_==NULL*

printf("Memory allocation error, new node could not be created");

**return**;

}

printf("INTRODUCE THE VALUE OF THE NEW NODE: **\n**");

printf("Give the name ");

scanf("%s", &in\_first->name);

printf("Give the street ");

scanf("%s", &in\_first->street);

printf("Give the city ");

scanf("%s", &in\_first->city);

printf("Estado: ");

scanf("%s", &in\_first->state);

printf("pin: ");

scanf("%i",&in\_first->pin);

**if**(first == NULL){

first = in\_first;

in\_first->before = NULL;

first ->next = NULL;

last = in\_first;

}

**else**{

in\_first->before = NULL;

in\_first->next = first;

first->before = in\_first;

first=in\_first;

}

contador++;

printf("**\n**THE NODE HAVE BEEN INTRODUCE CORRECTLY**\n\n**");

system("pause");

}

void insertlast()

{

Address \*in\_last = (Address \*) malloc(**sizeof**(Address));

**if**(!in\_last){*// new\_==NULL*

printf("Memory allocation error, new node could not be created");

**return**;

}

printf("INTRODUCE THE VALUE OF THE NEW NODE: **\n**");

printf("Give the name ");

scanf("%s", &in\_last->name);

printf("Give the street ");

scanf("%s", &in\_last->street);

printf("Give the city ");

scanf("%s", &in\_last->city);

printf("Estado: ");

scanf("%s", &in\_last->state);

printf("pin: ");

scanf("%i",&in\_last->pin);

**if**(last == NULL){

in\_last->before = NULL;

first = in\_last;

first ->next = NULL;

last = in\_last;

}

**else**{

last -> next = in\_last;

in\_last-> before = NULL;

in\_last -> next = NULL;

last = in\_last;

}

contador++;

printf("**\n**THE NODE HAVE BEEN INTRODUCE CORRECTLY**\n\n**");

system("pause");

}

void Anula()

{

Address \*actual = (Address \*) malloc(**sizeof**(Address));

actual = first;

Address\* before = (Address \*) malloc(**sizeof**(Address));

before = NULL;

**if**(first != NULL)

{

**while**(actual != NULL)

{

before = actual;

actual = actual ->next;

free(before);

contador--;

}

printf("**\n\n**LIST ELIMINATED SUCCESSFUL");

}

**else**

{

printf("**\n**THE LIST IS EMPTY**\n\n**");

}

printf("**\n**");

inicializa();

}

public:

void insertNode(int pos)

{

Address \*aux = nullptr;

aux = **new** Address;

**if** (pos == 0)

{

inserfirst();

}

**else** **if** (pos == tam())

{

insertlast();

}

**else** **if**(pos >= 1 && pos < tam())

{

printf("Give the name ");

scanf("%s", &aux->name);

printf("Give the street ");

scanf("%s", &aux->street);

printf("Give the city ");

scanf("%s", &aux->city);

printf("Estado: ");

scanf("%s", &aux->state);

printf("pin: ");

scanf("%i",&aux->pin);

Address \*aux2, \*aux3;

aux2 = first;

**for** (int i=0; i<pos; i++)

{

aux3 = aux2;

aux2 = aux2->next;

}

aux3->next = aux;

aux->next = aux2;

contador++;

printf("**\n**THE NODE HAVE BEEN INTRODUCE CORRECTLY**\n\n**");

system("pause");

}

}

void top()

{

Address \*temp = (Address \*) malloc(**sizeof**(Address));

temp = first;

**if** (first != NULL){

int i = 0;

printf("Actual %d", i);

printf("**\n** Name: %s**\n**", temp->name);

printf("street: %s**\n**", temp->street);

printf("City: %s**\n**", temp->city);

printf("State: %s**\n**", temp->state);

printf("pin: %i**\n\n**", temp->pin);

}

**else**{

printf("**\n** The list is empty**\n**");

}

}

*/\*void top()*

*{*

*Address \*temp = (Address \*) malloc(sizeof(Address));*

*temp = last;*

*if (first != NULL){*

*int i = contador;*

*printf("Actual %d", i-1);*

*printf("\n Name: %s\n", temp->name);*

*printf("street: %s\n", temp->street);*

*printf("City: %s\n", temp->city);*

*printf("State: %s\n", temp->state);*

*printf("pin: %i\n\n", temp->pin);*

*}*

*else{*

*printf("\n The list is empty\n");*

*}*

*}\*/*

void pop()

{

Address \*actual = (Address \*) malloc(**sizeof**(Address));

actual = first;

Address\* before = (Address \*) malloc(**sizeof**(Address));

before = NULL;

**if**(first != NULL){

printf("**\n** Name: %s**\n**", actual->name);

printf("street: %s**\n**", actual->street);

printf("City: %s**\n**", actual->city);

printf("State: %s**\n**", actual->state);

printf("pin: %i**\n\n**", actual->pin);

before = actual;

first = first ->next;

actual = actual ->next;

free(before);

contador--;

printf("**\n\n**NODE ELIMINATED SUCCESSFUL");

}

**else**{

printf("**\n**THE LIST IS EMPTY**\n\n**");

}

printf("**\n**");

}

*/\*void deque() //eliminar el ultimo nodo.*

*{*

*Address \*actual = (Address \*) malloc(sizeof(Address));*

*actual = first;*

*Address\* before = (Address \*) malloc(sizeof(Address));*

*before = NULL;*

*if(first != NULL){*

*while(actual != NULL){*

*if (actual == first && contador == 1){*

*printf("\n Name: %s\n", actual->name);*

*printf("street: %s\n", actual->street);*

*printf("City: %s\n", actual->city);*

*printf("State: %s\n", actual->state);*

*printf("pin: %i\n\n", actual->pin);*

*system("pause");*

*Anula();*

*return;*

*}*

*if (actual == last)*

*{*

*printf("\n Name: %s\n", actual->name);*

*printf("street: %s\n", actual->street);*

*printf("City: %s\n", actual->city);*

*printf("State: %s\n", actual->state);*

*printf("pin: %i\n\n", actual->pin);*

*last = before;*

*before -> next = actual -> next;*

*}*

*before = actual;*

*actual = actual ->next;*

*}*

*free(before);*

*contador--;*

*cout << "Tamaño :"<<contador<<endl;*

*}*

*else{*

*printf("\nTHE LIST IS EMPTY\n\n");*

*}*

*printf("\n");*

*}\*/*

void showreverse()

{

Address \*temp = (Address \*) malloc(**sizeof**(Address));

temp = last;

**if** (first != NULL){

int i = 0;

**while** (temp != NULL){

printf("Actual %d", i+1);

printf("**\n** Name: %s**\n**", temp->name);

printf("street: %s**\n**", temp->street);

printf("City: %s**\n**", temp->city);

printf("State: %s**\n**", temp->state);

printf("pin: %i**\n\n**", temp->pin);

temp = temp -> before;

i++;

}

}

**else**{

printf("**\n** The list is empty**\n**");

}

}

void getshow(){

**return** show();

}

void getseeknode(){

**return** seekNode();

}

void geteliminatenode(){

**return** eliminateNode();

}

void getinsertfirst(){

**return** inserfirst();

}

void getinsertlast(){

**return** insertlast();

}

void getLocaliza(){

**return** Localiza();

}

void getAnula(){

**return** Anula();

}

void getinicializa(){

**return** inicializa();

}

void gettop(){

**return** top();

}

void getpop(){

**return** pop();

}

*/\*void getdeque(){*

*return deque();*

*}\*/*

void getshowreverse(){

**return** showreverse();

}

};

void menu()

{

int opc;

List mi\_lista;

int pos;

mi\_lista.getinicializa();

**do**{

printf("**\t**INTRODUCE AN OPCION:**\n**");

printf("0.-Exit**\n**1.-Single list**\n**");

scanf("%d", &opc);

**switch**(opc){

**case** 0: printf("**\t**Goodbye");

system("pause");

**break**;

**case** 1: printf("**\t**Single list**\n\n**");

system("cls");

int opcion;

**do**{

printf("**\t**INTRODUCE AN OPCION:**\n**");

printf("0.-Exit**\n**1.-Show**\n**2.-Insert(pos)**\n**3.-Seek**\n**4.-Delete**\n**5.-Insert first(Push)**\n**6.-Insert last(queue)**\n**7.-Localiza**\n**8.-Anula**\n**");

printf("**\n**9.-pop**\n**10.-top**\n**11.-deque**\n**12.-Front**\n**13.-showreverse**\n**");

scanf("%d", &opcion);

**switch**(opcion){

**case** 0: printf("**\t** Adios**\n**");

system("pause");

**break**;

**case** 1: printf("**\t**Show the list**\n\n**");

system("cls");

mi\_lista.getshow();

system("pause");

**break**;

**case** 2: system("cls");

printf("**\t**Insert a node in the list**\n\n**");

printf("Posicion a insertar:**\n**");

scanf("%d", &pos);

mi\_lista.insertNode(pos);

**break**;

**case** 3: system("cls");

printf("**\t**Seek a node in the list**\n\n**");

mi\_lista.getseeknode();

**break**;

**case** 4: system("cls");

printf("**\t**Delete node**\n\n**");

mi\_lista.geteliminatenode();

**break**;

**case** 5: system("cls");

printf("**\t**Insert first**\n\n**");

mi\_lista.getinsertfirst();

**break**;

**case** 6: system("cls");

printf("**\t**Insert last**\n\n**");

mi\_lista.getinsertlast();

**break**;

**case** 7: system("cls");

printf("**\t**Localiza**\n\n**");

mi\_lista.getLocaliza();

**break**;

**case** 8: system("cls");

printf("**\t**Anula**\n\n**");

mi\_lista.getAnula();

**break**;

**case** 9: system("cls");

printf("**\t**POP**\n\n**");

mi\_lista.getpop();

**break**;

**case** 10: system("cls");

printf("**\t**TOP**\n\n**");

mi\_lista.gettop();

**break**;

**case** 11: system("cls");

printf("**\t**Deque**\n\n**");

mi\_lista.getpop();

**break**;

**case** 12: system("cls");

printf("**\t**TOP**\n\n**");

mi\_lista.gettop();

**break**;

**case** 13: system("cls");

printf("**\t**Show Reverse**\n\n**");

mi\_lista.getshowreverse();

**break**;

}

}**while**(opcion!=0);

**break**;

default: printf("**\t**Choose a correct value");

**break**;

}

system("cls");

}**while**(opc!=0);

}

int main()

{

menu();

**return** 0;

}