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.

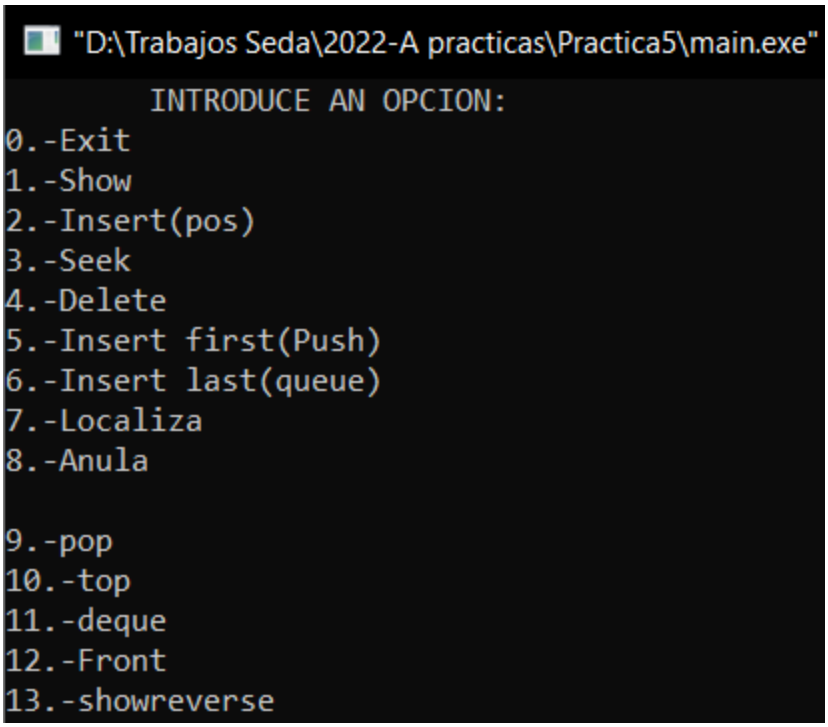
Nombre alumno: Padilla Pérez Jorge Daray

Nombre profesor: Julio Esteban Valdes Lopez

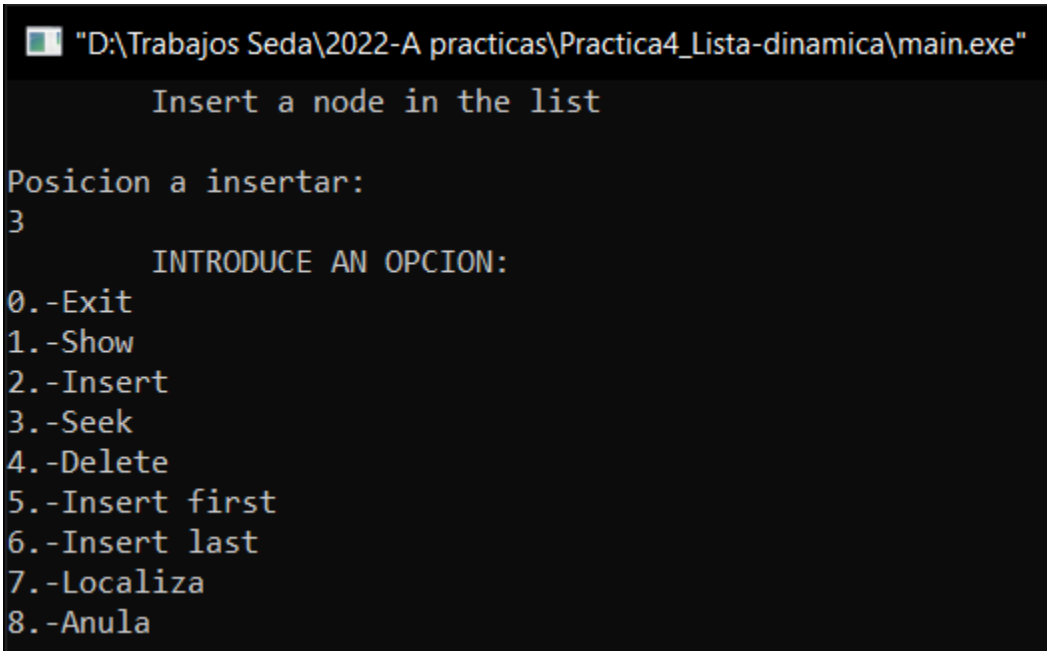
## Introducción

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.

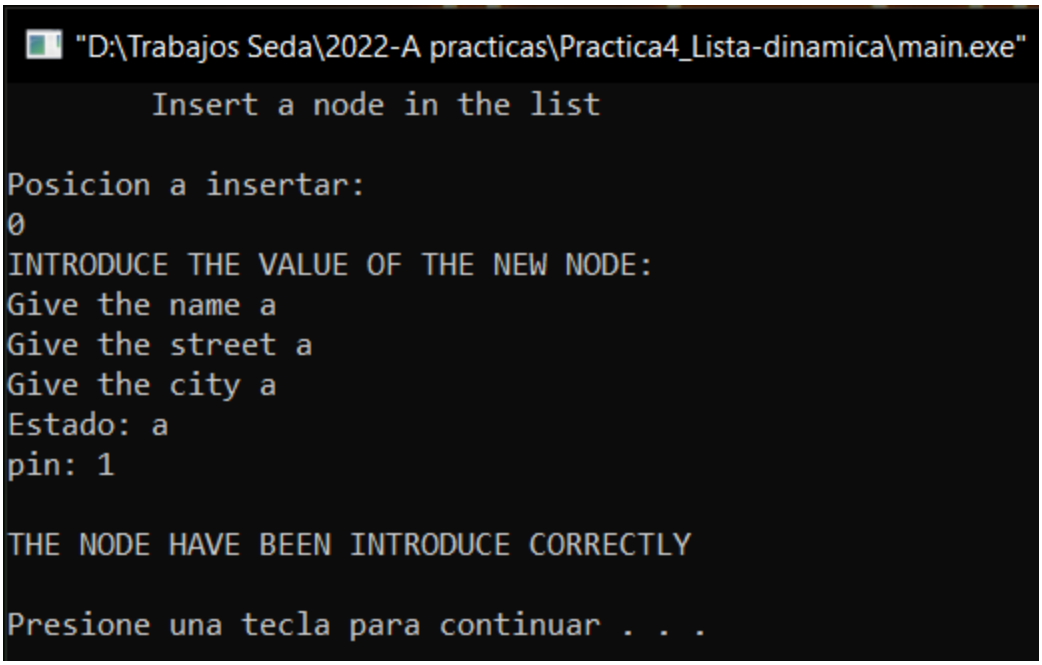
El programa está hecho con clases protegidas utilizando herencia y polimorfismo para usar correctamente la clase lista, en la cual se incluye los métodos básicos de una lista, además de tener los métodos de la pila y cola anteriormente mencionadas.



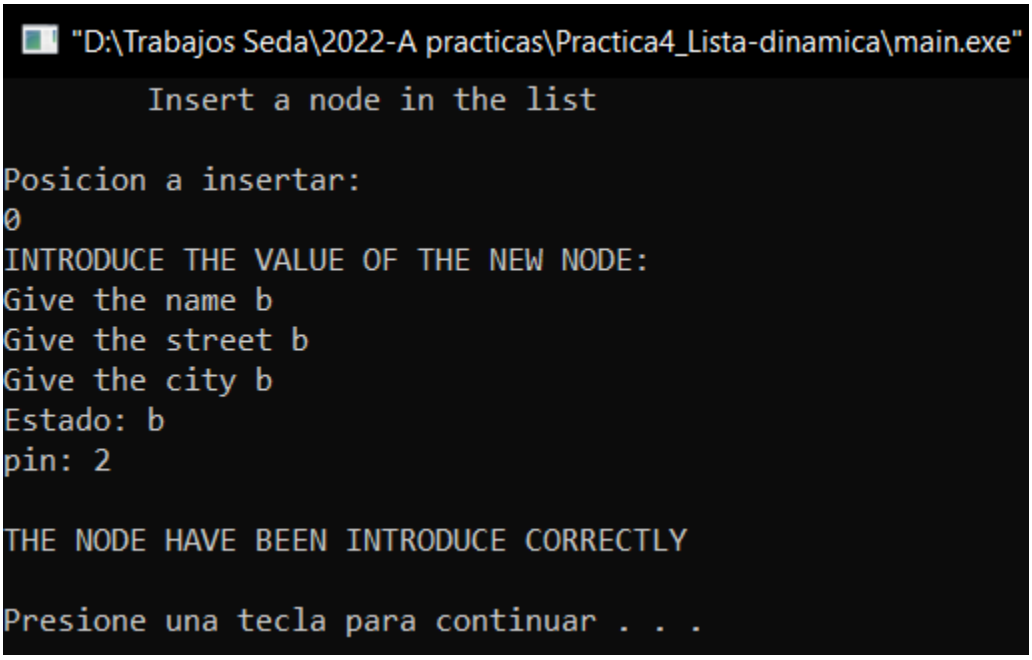

Aquí se aprecia el menú.



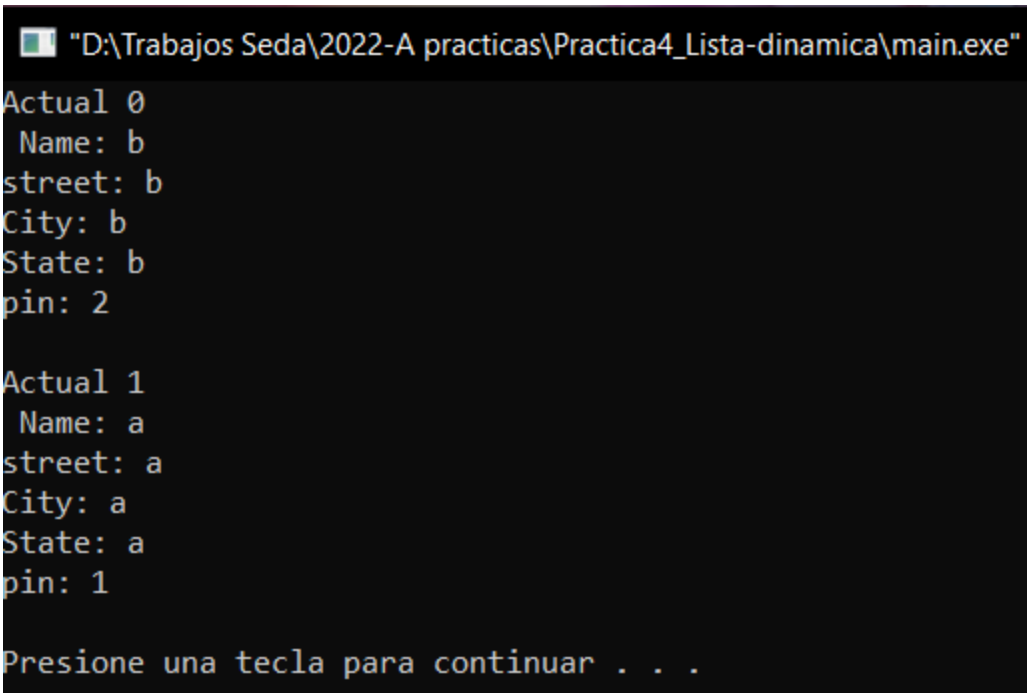

Si queremos insertar en una posición invalida no nos deja, ya que tiene que ser lineal y en este caso debería empezar en la posición 0.



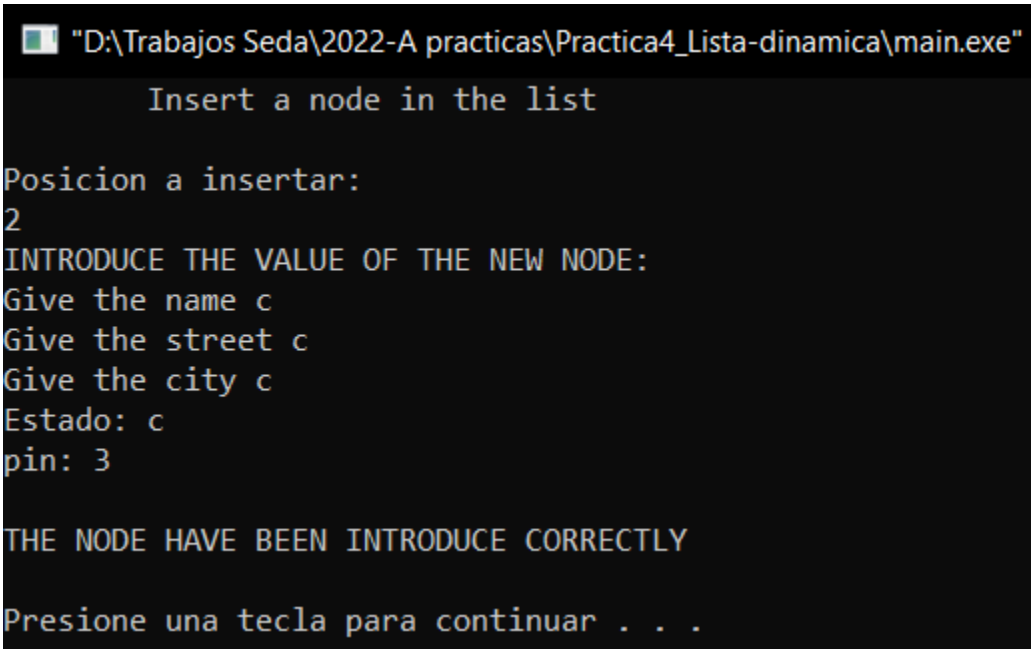

Aquí insertamos un nodo en la posición 0.



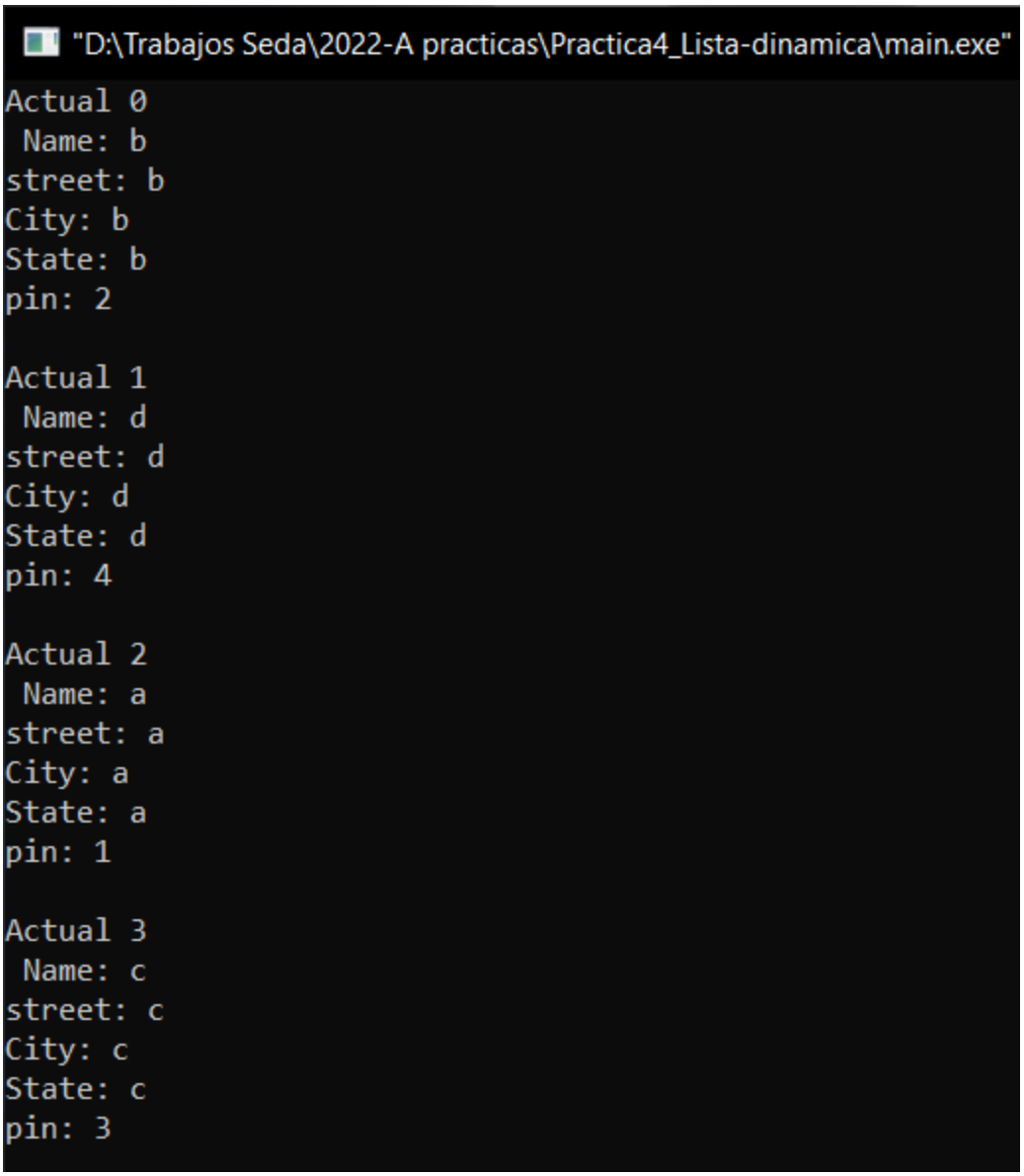

Aquí insertamos otro nodo en la posición 0, ósea al principio de la lista.



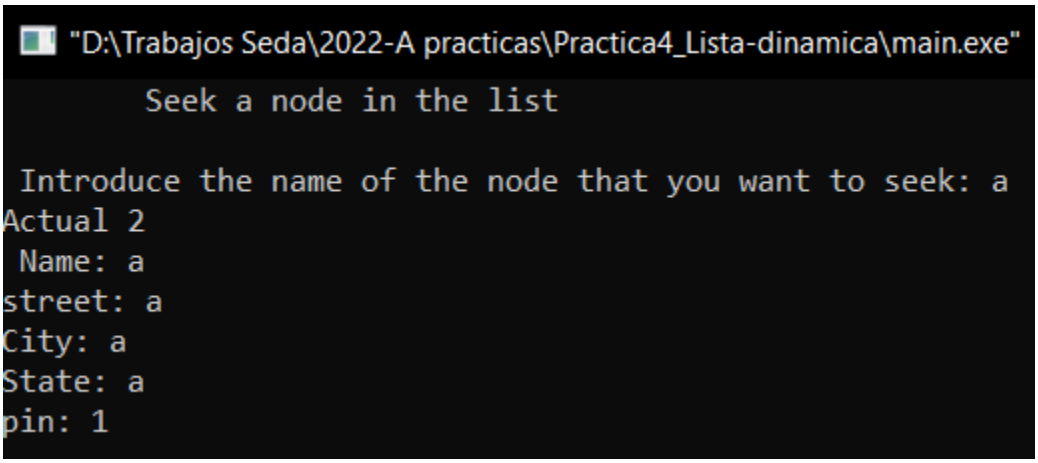

Aquí mostramos la lista.



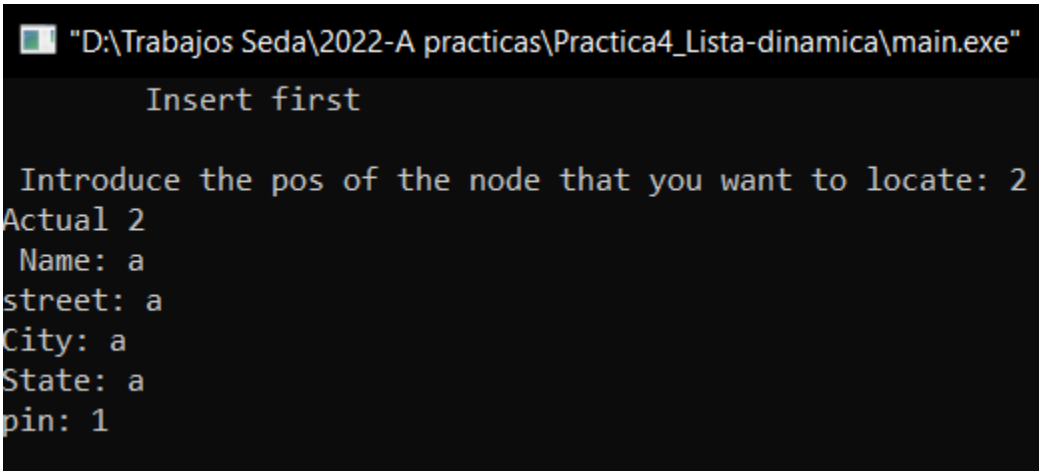

Insertamos un tercer nodo en la ultima posición.



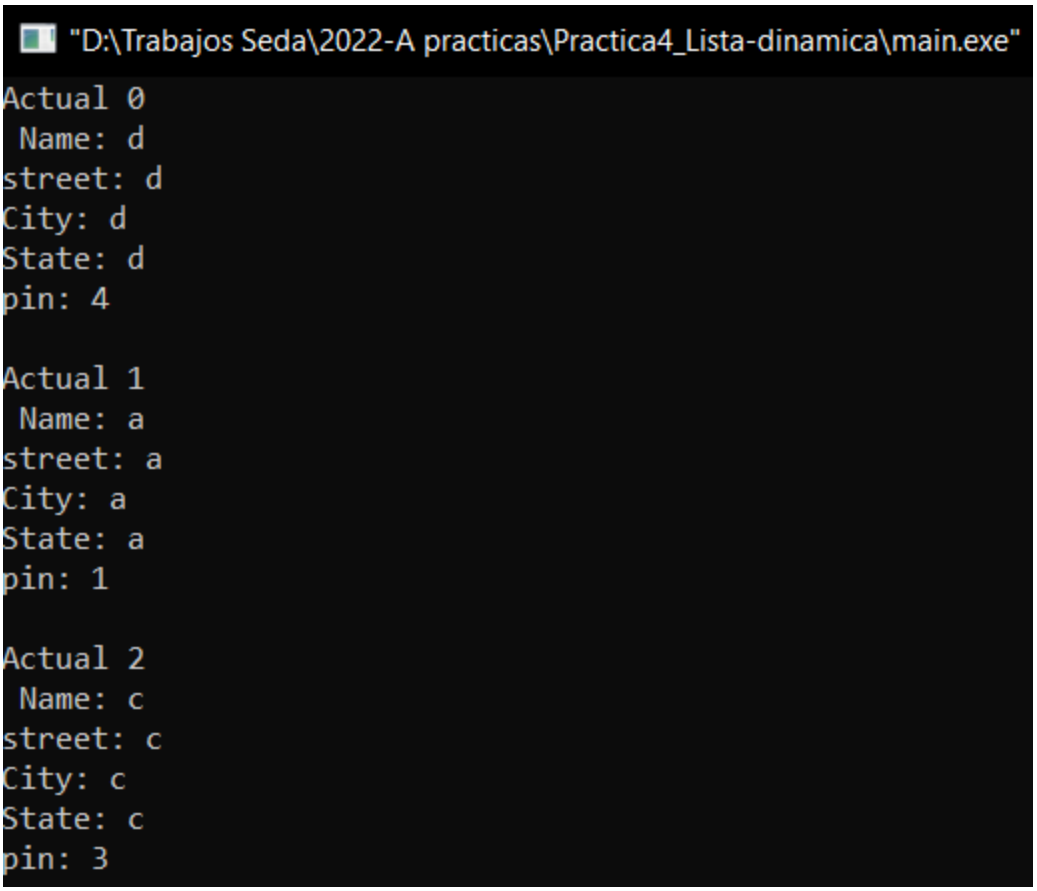

Insertamos un nodo 4 en la posición 1 por lo que recorre a los demás.



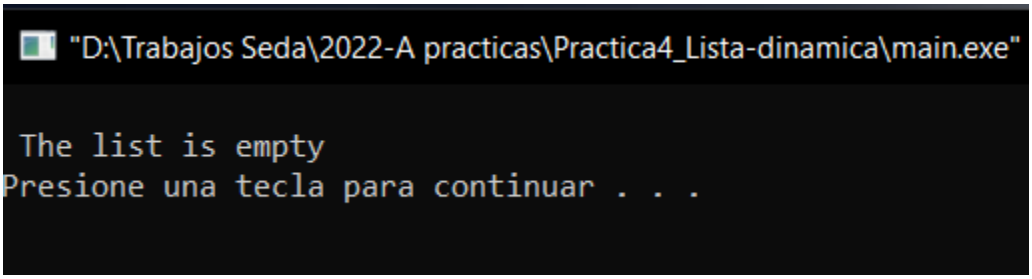

Aquí usamos la función recupera que en este caso pide el nombre del nodo.




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




Aquí eliminamos el pin 2.




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("\nThe link of the node have been eliminated");
                found = 1;
            }
            before = actual;
```

```
            actual = actual ->next;
        }
        if(found == 0){
            printf("\nthe node was not found\n\n");
        }
        else{
            free(before);
            contador--;
            printf("\n\nNODE ELIMINATED SUCCESSFUL");
        }
    }
    else{
        printf("\nTHE 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("\nTHE 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("\nTHE 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\nLIST ELIMINATED SUCCESSFUL");
```

```
        }
        else
        {
        printf("\nTHE 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("\nTHE 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\nNODE ELIMINATED SUCCESSFUL");
    }
```

```
    else{
        printf("\nTHE 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("\tINTRODUCE AN OPCION:\n");
    printf("0.-Exit\n1.-Single list\n");
    scanf("%d", &opc);


    switch(opc){
        case 0: printf("\tGoodbye");
                system("pause");
                break;
        case 1: printf("\tSingle list\n\n");
                system("cls");
                int opcion;
                do{
                printf("\tINTRODUCE AN OPCION:\n");
                printf("0.-Exit\n1.-Show\n2.-Insert(pos)\n3.-Seek\n4.-
Delete\n5.-Insert first(Push)\n6.-Insert last(queue)\n7.-Localiza\n8.-
Anula\n");
                printf("\n9.-pop\n10.-top\n11.-deque\n12.-Front\n13.-
showreverse\n");
                scanf("%d", &opcion);
                    switch(opcion){
                        case 0: printf("\t Adios\n");
                            system("pause");
                            break;
                        case 1: printf("\tShow the list\n\n");
                            system("cls");
                            mi_lista.getshow();
                            system("pause");
                            break;
                        case 2: system("cls");
                            printf("\tInsert 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("\tSeek a node in the list\n\n");
                            mi_lista.getseeknode();
                            break;
```

```
                        case 4: system("cls");
                            printf("\tDelete node\n\n");
                            mi_lista.geteliminatenode();
                            break;
                        case 5: system("cls");
                            printf("\tInsert first\n\n");
                            mi_lista.getinsertfirst();
                            break;
                        case 6: system("cls");
                            printf("\tInsert last\n\n");
                            mi_lista.getinsertlast();
                            break;
                        case 7: system("cls");
                            printf("\tLocaliza\n\n");
                            mi_lista.getLocaliza();
                            break;
                        case 8: system("cls");
                            printf("\tAnula\n\n");
                            mi_lista.getAnula();
                            break;
                        case 9: system("cls");
                            printf("\tPOP\n\n");
                            mi_lista.getpop();
                            break;
                        case 10: system("cls");
                            printf("\tTOP\n\n");
                            mi_lista.gettop();
                            break;
                        case 11: system("cls");
                            printf("\tDeque\n\n");
                            mi_lista.getpop();
                            break;
                        case 12: system("cls");
                            printf("\tTOP\n\n");
                            mi_lista.gettop();
                            break;
                        case 13: system("cls");
                            printf("\tShow Reverse\n\n");
                            mi_lista.getshowreverse();
                            break;
                    }
                }while(opcion!=0);
                break;
            default: printf("\tChoose a correct value");
                break;
        }
        system("cls");
        }while(opc!=0);

}

int main()
{
```

```
    menu();
    return 0;
}
```