

15-11-2021



# Lista dinámica simple con encabezado

Practica: 6

Materia: Estructura de datos

Sección: D01.

Código: 216584703

Carrera: Ingeniería en computación.

Nombre alumno: Padilla Pérez Jorge  
Daray

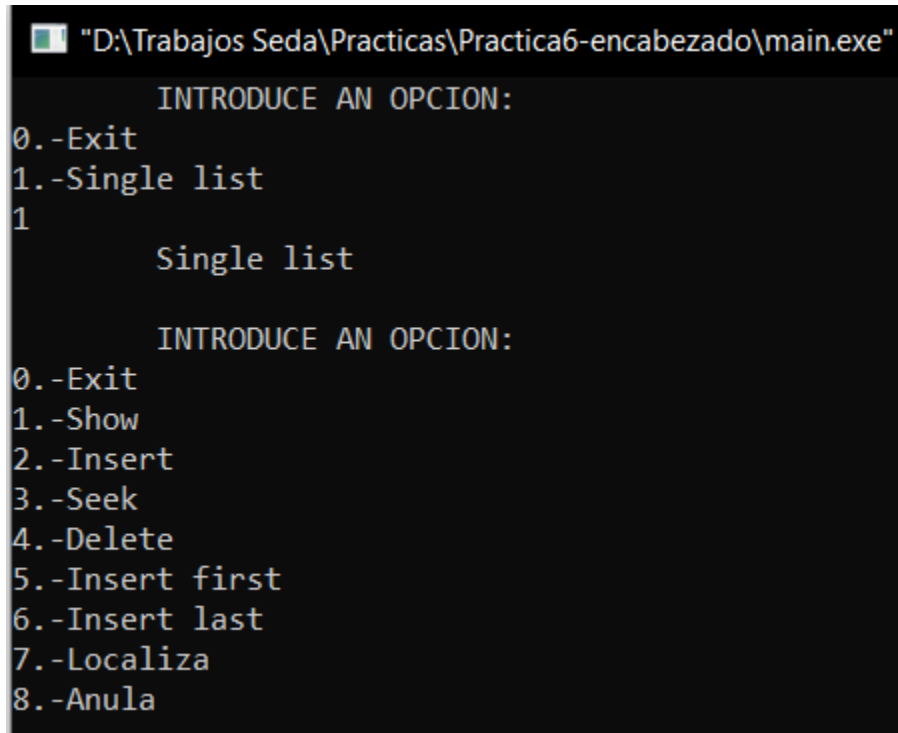
Nombre profesor: Julio Esteban  
Valdes Lopez

# Introducción

Mi práctica consiste en la implementación de una lista dinámica en la cual se implementan lo típico de un TDA Lista lo que viene siendo insertar en orden como si fuera una lista consultar elemento remove etc. En esta practica se logro hacer todo lo pedido para esta, además de poder realizar la función de localizar posición por elemento la cual me costó más, pero si se logró.

También aunque no se implementaron las funciones anterior y siguiente creo que es una función fácil de comprender y de hacer por lo que me parece bien que se hayan saltado esas 2 funciones que en si no existe una utilidad buena para ellos.

# Pantallazos



```
"D:\Trabajos Seda\Practicas\Practica6-encabezado\main.exe"

    INTRODUCE AN OPCION:
0.-Exit
1.-Single list
1
    Single list

    INTRODUCE AN OPCION:
0.-Exit
1.-Show
2.-Insert
3.-Seek
4.-Delete
5.-Insert first
6.-Insert last
7.-Localiza
8.-Anula
```

El menú de la lista.

```
"D:\Trabajos Seda\Practicas\Practica6-encabezado\main.exe"
4.-Delete
5.-Insert first
6.-Insert last
7.-Localiza
8.-Anula
1
    Show the list

Actual 0
  Name: c
street: c
City: c
State: c
pin: 30

Actual 1
  Name: b
street: b
City: b
State: b
pin: 20

Actual 2
  Name: a
street: a
City: a
State: a
pin: 10
```

Se insertan 3 nodos con los diferentes métodos.

```
"D:\Trabajos Seda\Practicas\Practica6-encabezado\main.exe"
1.-Show
2.-Insert
3.-Seek
4.-Delete
5.-Insert first
6.-Insert last
7.-Localiza
8.-Anula
3
    Seek a node in the list

    Introduce the name of the node that you want to seek: a
Actual 2
    Name: a
    street: a
    City: a
    State: a
    pin: 10
```

Función recupera la posición con el dato en este caso el nombre a.

```
"D:\Trabajos Seda\Practicas\Practica6-encabezado\main.exe"
1.-Show
2.-Insert
3.-Seek
4.-Delete
5.-Insert first
6.-Insert last
7.-Localiza
8.-Anula
7
    Insert first

    Introduce the pos of the node that you want to locate: 2
Actual 2
    Name: a
street: a
City: a
State: a
pin: 10
```

Aquí se aprecia la función localizar el dato por la posición.

```
"D:\Trabajos Seda\Practicas\Practica6-encabezado\main.exe"
State: a
pin: 10

      INTRODUCE AN OPCION:
0.-Exit
1.-Show
2.-Insert
3.-Seek
4.-Delete
5.-Insert first
6.-Insert last
7.-Localiza
8.-Anula
8
      Insert first

NODE ELIMINATED SUCCESSFUL
      INTRODUCE AN OPCION:
0.-Exit
1.-Show
2.-Insert
3.-Seek
4.-Delete
5.-Insert first
6.-Insert last
7.-Localiza
8.-Anula
```

Aquí se anula la lista y ya no hay nodos la lista esta vacía.

# Conclusión

Respecto a la realización del código concluyo que se logró bien la realización de este programa ya que a lo que se puede apreciar funciona de manera correcta como debería hacerlo una lista además de implementar bien sus funciones recalco mis conocimientos, y estoy abierto a posibles errores que pueda tener el programa al momento de que el profe la evalúe.

También se concluye que aunque no parezca que tiene una utilidad buena al momento de referirse a trabajo si lo piensas un rato le hayas muchas utilidades importantes tanto que pensándolo muchas aplicaciones y juegos realizan una lista implementada en estos mismos por lo cual espero poder aprender mas sobre las listas y los tipos de datos abstractos



# Código fuente

```
#include <iostream>

#include<stdlib.h>

#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 *ant;
    }Address;
    Address * first = nullptr;
    Address * last = nullptr;
    int contador = 0;

    void show()
    {
        Address *temp = (Address *) malloc(sizeof(Address));
        temp = first;
        if (first != NULL){
            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{  
                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");
}

};

class List: StructBase
{
protected:
    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: ");
        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;
    first ->next = NULL;
    last = in_first;
}
else{
    in_first->next = 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: ");
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){
        first = in_last;
        first ->next = NULL;
        last = in_last;
    }
    else{
        last -> next = in_last;
        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)
        {

            if(actual == first)
            {
                first = first ->next;
            }
            else
            {
                before -> next = actual -> next;
            }
            before = actual;
            actual = actual ->next;
        }
        free(before);
        printf("\n\nNODE ELIMINATED SUCCESSFUL");
    }
    else
    {
        printf("\nTHE LIST IS EMPTY\n\n");
    }
    printf("\n");
    contador = 0;
}

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++;
}
system("pause");
}
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();
}
int gettam(){
    return tam();
}

```

```

        int getcont(){
            return contador;
        }

};

void menu()
{
    int opc;
    List mi_lista;
    int pos;

    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");
                    int opcion;
                    do{
                        printf("\tINTRODUCE AN OPCION:\n");
                        printf("0.-Exit\n1.-Show\n2.-Insert\n3.-Seek\n4.-Delete\n5.-
Insert first\n6.-Insert last\n7.-Localiza\n8.-Anula\n");
                        scanf("%d", &opcion);
                        switch(opcion){

```

```
case 0: printf("\t Adios\n");
        system("pause");
        break;
case 1: printf("\tShow the list\n\n");
        mi_lista.getshow();
        mi_lista.getcont();
        system("pause");
        break;
case 2: printf("\tInsert a node in the list\n\n");
        printf("Posicion a insertar:\n");
        scanf("%d", &pos);
        mi_lista.insertNode(pos);
        break;
case 3: printf("\tSeek a node in the list\n\n");
        mi_lista.getseeknode();
        break;
case 4: printf("\tDelete node\n\n");
        mi_lista.geteliminatenode();
        break;
case 5: printf("\tInsert first\n\n");
        mi_lista.getinsertfirst();
        break;
case 6: printf("\tInsert first\n\n");
        mi_lista.getinsertlast();
        break;
case 7: printf("\tInsert first\n\n");
        mi_lista.getLocaliza();
        break;
case 8: printf("\tInsert first\n\n");
        mi_lista.getAnula();
```

```

                break;
            }
        }while(opcion!=0);
        break;
    default: printf("\tChoose a correct value");
        break;
}
system("cls");
}while(opc!=0);

}

int main()
{
    menu();
    return 0;
}

```