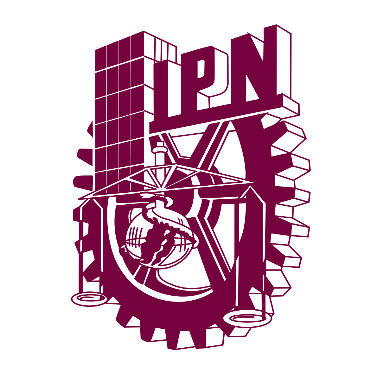
****

**Instituto Politécnico Nacional**

**Escuela Superior de Computo**

*Alumno:*

* *Monroy Ramírez Oscar G.*

***Grupo:*** *2CM23*

***Unidad de Aprendizaje:*** *Algoritmos y Estructuras de Datos*

***Evidencia:*** *Programa renglón No. 1*

***Docente:*** *De Luna Caballero Roberto*

***Fecha****: lunes 9 de noviembre de 2020*

Contenido

[Introducción 3](#_Toc55851250)

[Marco Teórico 4](#_Toc55851251)

[Definición de Memoria dinámica 4](#_Toc55851252)

[Definición de Memoria Estatica 4](#_Toc55851253)

[Diferencias, ventajas y desventajas 4](#_Toc55851254)

[Funciones correspondientes a la Memoria Dinamica, con ejemplos simples. 5](#_Toc55851255)

[MALLOC 5](#_Toc55851256)

[CALLOC 6](#_Toc55851257)

[REALLOC 7](#_Toc55851258)

[FREE O CFREE 8](#_Toc55851259)

[Abstracción y tipo de dato abstracto 8](#_Toc55851260)

[-Tipo de Dato Abstracto 9](#_Toc55851261)

[Programa 13](#_Toc55851262)

[Renglón No.1 13](#_Toc55851263)

[Explicación del programa 27](#_Toc55851264)

[Conclusiones 27](#_Toc55851265)

[Referencias Bibliográficas: 28](#_Toc55851266)

# 

# Introducción

En muchas ocasiones como programadores, nos podemos encontrar con diversos problemas, después de todo es parte fundamental darles solución, muchas veces son problemas los cuales son fáciles de resolver, y en otras ocasiones hay problemas más difíciles, en esta ocasión Veremos el uso de la memoria dinámica.

Muchas veces cuándo del uso de memoria se habla, es muy importante considerar que en muchos de los casos no sabemos cuánta memoria utilizaremos en el transcurso del programa, en el buen uso de la programación debemos considerar que el no saber cuánta memoria utilizaremos, debemos pedirle al sistema que pida la memoria necesaria mientras ejecuta nuestro programa, por ello usamos algunas funciones especiales para pedirle al sistema más memoria mientras ésta se requiere, una estas funciones nos permite asignar la memoria correspondiente y que requeriremos, y otra se encargará específicamente liberar aquella memoria que nosotros hemos solicitado como a continuación se explicara.

Este ejemplo usaremos la memoria dinámica para poder manejar un arreglo de números y en caso de que se requiera más definido por el usuario, este punto la agrandar su arreglo a beneficio y a su consideración, o en su caso eliminarlo.

Y es que aunque sea difícil implementar en el desarrollo de un programa o aplicación, las funciones de memoria dinámica, inclusive sea difícil en estructuras complejas como lo son árboles, grafos, entre otras (pues necesitamos una forma de para solicitar y liberar memoria para nuevas variables que puedan ser necesarios durante la ejecución de nuestros programas), al igual que la memoria dinámica pudiese afectar el rendimiento al no conocer el tamaño de las variables, y de igual forma si tengo que llevar muchas más tareas a cabo para la ejecución correcta de este tipo de operaciones, no podemos dejar de ver que dentro de las grandes ventajas tenemos, qué podemos disponer de un espacio de memoria arbitrario, y que podemos ir incrementando durante la ejecución del programa permitiendo trabajar con arreglos dinámicos, al igual que podemos variar su tamaño durante la ejecución del mismo programa para después ser liberado.

Por lo anterior, la memoria dinámica no es más que una herramienta, que a manos de un buen programador puede ser de gran ayuda, pero el uso innecesario, no causaría más que enredos y trabajos innecesarios, qué afectarían el rendimiento del ordenador y el tiempo de programación. Por ello la importancia discernir al momento de establecer el programa que queremos realizar qué me conviene más, si el uso de la gran memoria estática, uh la memoria dinámica.

Ahora la memoria dinámica es mejor usarla para grandes cantidades de datos, para cuando retiramos modificar el tamaño desierto arreglo o estructura en el tiempo de ejecutar nuestro programa, no se desperdicia memoria que no se utilicen su totalidad, tenemos un menor costo por bit y un menor consumo de potencia (si es que se usa correctamente).

Una vez mencionadas algunas de las ventajas y desventajas del uso de las respectivas memorias mencionadas es momento de pasar observar de cerca lo que es la memoria dinámica y cómo hacer uso de ella adecuadamente mediante las funciones correspondientes.

# Marco Teórico

## Definición de Memoria dinámica

Según Kernighan (1978) es memoria que se reserva en tiempo de ejecución. Su principal ventaja frente a la estática es que su tamaño puede variar durante la ejecución del programa. (En C, el programador es encargado de liberar esta memoria cuando no la utilice más). El uso de memoria dinámica es necesario cuando no se sabe el número exacto de datos/elementos a tratar.

## Definición de Memoria Estatica

Según Kernighan (1978) es el espacio en memoria que se crea al declarar variables de cualquier tipo de dato (primitivas [int,char...] o derivados [struct,matrices,punteros...]). La memoria que estas variables ocupan no puede cambiarse durante la ejecución y tampoco puede ser liberada manualmente.

## Diferencias, ventajas y desventajas

Otra ventaja de la memoria dinámica es que se puede ir incrementando durante la ejecución del programa. Esto permite, por ejemplo, trabajar con arreglos dinámicos. Aunque en C, a partir del estándar C99 se permite la creación de arreglos cuyo tamaño se determina en tiempo de ejecución, no todos los compiladores implementan este estándar. Además, se sigue teniendo la limitante de que su tamaño no puede cambiar una vez que se especifica, cosa que sí se puede lograr asignando memoria de forma dinámica.

Una desventaja de la memoria dinámica es que es más difícil de manejar. La memoria estática tiene una duración fija, que se reserva y libera de forma automática. En contraste, la memoria dinámica se reserva de forma explícita y continúa existiendo hasta que sea liberada, generalmente por parte del programador.

La memoria dinámica puede afectar el rendimiento. Puesto que con la memoria estática el tamaño de las variables se conoce en tiempo de compilación, esta información está incluida en el código objeto generado, por lo cual el proceso es muy eficiente. Cuando se reserva memoria de manera dinámica, se tienen que llevar a cabo varias tareas, como buscar un bloque de memoria libre y almacenar la posición y tamaño de la memoria asignada, de manera que pueda ser liberada más adelante. Todo esto representa una carga adicional, aunque esto depende de la implementación y hay técnicas para reducir su impacto.

## Funciones correspondientes a la Memoria Dinamica, con ejemplos simples.

### MALLOC

La función malloc reserva un bloque de memoria y devuelve un puntero void al inicio de la misma. Tiene la siguiente definición:

void \*malloc(size\_t size);

donde el parámetro size especifica el número de bytes a reservar. En caso de que no se pueda realizar la asignación, devuelve el valor nulo (definido en la macro NULL), lo que permite saber si hubo errores en la asignación de memoria.

Ej:

int \*puntero;

char \*puntcarc;

puntero=(int \*)malloc(4);

puntcarc=(char \*)malloc(200);

A continuación ejemplos de uso:

int \*i;

…

*/\* Reservamos la memoria suficiente para almacenar un int y asignamos su dirección a i \*/*

i = malloc(**sizeof**(int));

*/\* Verificamos que la asignación se haya realizado correctamente \*/*

**if** (i == NULL) {

*/\* Error al intentar reservar memoria \*/*

}

Uno de los usos más comunes de la memoria dinámica es la creación de vectores cuyo número de elementos se define en tiempo de ejecución:

int \*vect1, n;

printf("N£mero de elementos del vector: ");

scanf("%d", &n);

*/\* reservar memoria para almacenar n enteros \*/*

vect1 = malloc(n \* **sizeof**(int));

*/\* Verificamos que la asignación se haya realizado correctamente \*/*

**if** (vect1 == NULL) {

*/\* Error al intentar reservar memoria \*/*

}

### CALLOC

La función calloc funciona de modo similar a malloc, pero además de reservar memoria, inicializa a 0 la memoria reservada. Se usa comúnmente para arreglos y matrices. Está definida de esta forma:

void \*calloc(size\_t nmemb, size\_t size);

El parámetro nmemb indica el número de elementos a reservar, y size el tamaño de cada elemento. El ejemplo anterior se podría reescribir con calloc de esta forma:

int \*vect1, n;

printf("N£mero de elementos del vector: ");

scanf("%d", &n);

*/\* Reservar memoria para almacenar n enteros \*/*

vect1 = calloc(n, **sizeof**(int));

*/\* Verificamos que la asignación se haya realizado correctamente \*/*

**if** (vect1 == NULL) {

*/\* Error al intentar reservar memoria \*/*

}

### REALLOC

La función realloc redimensiona el espacio asignado de forma dinámica anteriormente a un puntero. Tiene la siguiente definición:

void \*realloc(void \*ptr, size\_t size);

Donde ptr es el puntero a redimensionar, y size el nuevo tamaño, en bytes, que tendrá. Si el puntero que se le pasa tiene el valor nulo, esta función actúa como malloc. Si la reasignación no se pudo hacer con éxito, devuelve un puntero nulo, dejando intacto el puntero que se pasa por parámetro. Al usar realloc, se debería usar un puntero temporal. De lo contrario, podríamos tener una fuga de memoria, si es que ocurriera un error en realloc.

Ejemplo de realloc usando puntero temporal:

*/\* Reservamos 5 bytes \*/*

void \*ptr = malloc(5);

…

*/\* Redimensionamos el puntero (a 10 bytes) y lo asignamos a un puntero temporal \*/*

void \*tmp\_ptr = realloc(ptr, 10);

**if** (tmp\_ptr == NULL) {

*/\* Error: tomar medidas necesarias \*/*

}

**else** {

*/\* Reasignación exitosa. Asignar memoria a ptr \*/*

ptr = tmp\_ptr;

}

Cuando se redimensiona la memoria con realloc, si el nuevo tamaño (parámetro size) es mayor que el anterior, se conservan todos los valores originales, quedando los bytes restantes sin inicializar. Si el nuevo tamaño es menor, se conservan los valores de los primeros size bytes. Los restantes también se dejan intactos, pero no son parte del bloque regresado por la función.

### FREE O CFREE

La función free sirve para liberar memoria que se asignó dinámicamente. Si el puntero es nulo, free no hace nada. Además existe la función cfree, que sirve para liberar memoria de los elementos que han sido reservados con calloc(). Tienen la siguiente definición:

void free(void \*ptr); void cfree (void \*ptr);

El parámetro ptr es el puntero a la memoria que se desea liberar:

int \*i;

i = malloc(**sizeof**(int));

…

free(i);

Una vez liberada la memoria, si se quiere volver a utilizar el puntero, primero se debe reservar nueva memoria con malloc o calloc:

int \*i = malloc(**sizeof**(int));

…

free(i);

*/\* Reutilizamos i, ahora para reservar memoria para dos enteros \*/*

i = malloc(2 \* **sizeof**(int));

…

*/\* Volvemos a liberar la memoria cuando ya no la necesitamos \*/*

free(i);

### Abstracción y tipo de dato abstracto

Según Sirus (2020) un Tipo de dato abstracto  (TDA) es un conjunto de datos u objetos al cual se le asocian operaciones. El TDA provee de una interfaz con la cual es posible realizar las operaciones permitidas, abstrayéndose de la manera en como estén implementadas dichas operaciones. Esto quiere decir que un mismo TDA puede ser implementado utilizando distintas estructuras de datos y proveer la misma funcionalidad.**[5]**

El paradigma de orientación a objetos permite el encapsulamiento de los datos y las operaciones mediante la definición de clases e interfaces, lo cual permite ocultar la manera en cómo ha sido implementado el TDA y solo permite el acceso a los datos a través de las operaciones provistas por la interfaz. (Sirus, 2020) **[5]**

## -Tipo de Dato Abstracto

##### -Pila

Una pila es una colección ordenada de elementos en la que pueden insertarse y suprimirse por un extremo, llamado ***tope,*** nuevos elementos. Se puede representar una pila como en la figura No. 6 (“Representacion de pila”).

A diferencia del arreglo, la definición de pila incorpora la inserción y supresión de elementos, de tal manera que esta es un objeto dinámico constantemente variable. En consecuencia, surge una pregunta ¿Cómo cambia la pila? La definición especifica que un extremo de la fila se designa como el tope de la misma. Pueden agregarse nuevos elementos en el tope de la pila (en cuyo caso éste se mueve hacia arriba para corresponder al nuevo elemento que ocupa la cima), o pueden quitarse los elementos que están en el tope (y entonces el tope se mueve hacia abajo para corresponder al nuevo elemento que se encuentra hacia arriba). ¿Para responder la pregunta cuál camino es hacia arriba? hj tenemos que definir a 1 de los extremos de la pila como su tope; es decir, en qué extremo pueden colocar suprimirse elementos. En la figura 6 se observa que el hecho que la F esté físicamente más arriba que el resto de los elementos de la pila, implica que es el elemento tope actual. Sí se agregan nuevos elementos a la pila, estos deben colocarse encima de F, y si se suprimen algunos, F será el primero en ser borrado. Esto sí indica también por las líneas verticales, las cuales se extienden más allá de los elementos de la pila en dirección del tope de la misma.

|  |
| --- |
| F |
| A |
| B |
| G |

**“Representación de Pila” [Elaboración propia]**

La representación de una pila como un tipo de dato abstracto es directa. Usamos ***eltype*** Para denotar el tipo del elemento de la pila y parametrizamos el tipo de pila con ***eltype.***

**abstract tydef** <<eltype>> STACK (eltype);

**abstract** empty(s)

STACK (eltype)s;

postcondition empty==(len(s)==0);

**abstract** eltype pop(s)

*STACK* (eltype) s;

precondition empty==FALSE;

**postcondition** pop == first(s’);

s==sub(s’, 1, len(s’)-1);

***abstract*** *push (s, elt)*

*STACK (eltype) s;*

*eltype elt;*

***postcondition*** *s==<elt>+s’*

Antes de programar la solución de un problema que usa una pila, debe decidirse como representar una pila mediante las estructuras de datos existentes en nuestro lenguaje de programación. Cómo se verá, hay varias maneras de representar una pila en lenguaje C. Considere C ahora la más simple:

Una pila es una colección ordenada de elementos, el lenguaje se incluye un tipo de datos que es una colección ordenada de elementos: el arreglo. Por ello, excepto por comenzar un programa con la declaración de una variable *pila* cómo arreglo. siempre que la solución de un problema requiere el uso de una pila. Sin embargo, una pila y un arreglo son dos cosas por completo diferentes. El número de elementos de un arreglo es fijo y se asigna mediante su declaración. Y en general, el usuario no puede cambiar dicho número. Por otra parte, una pila es lo fundamental un objeto dinámico cuyo tamaño cambia constantemente tanto se le agreguen o quiten elementos.

Sin embargo, aunque una regla no puede ser una pila, puede usarse como tal. Es decir, puede declararse un arreglo lo bastante grande para admitir el tamaño máximo de la pila. Durante la ejecución del programa, la pila puede crecer y contraerse dentro de su espacio reservado. El fondo fijo de la pila es un extremo del arreglo mientras que su tope cambia en forma constante cuando se agregan o quitan elementos. Así, es necesario otro campo para que en cada paso de la ejecución del programa, registre la posición actual del elemento tope de la pila.

En consecuencia se puede declarar una pila el lenguaje C, com una estructura que contiene dos datos: un arreglo para guardar los elementos de la pila y un entero para indicar la posición del elemento tope actual dentro del arreglo; lo cual puede hacerse para una pila de enteros por medios de las declaraciones:

**#define** STACKSIZE 100

STRUCT stack {

**int** top;

**int** items[STACKSIZE];

};

Una vez hecho puede declararse una pila S,por medio de:

**Struct** stack s;

##### -Colas

La **cola** es una colección ordenada de elementos de la que se pueden borrar elementos en un extremo (llamado **frente** de la cola) o insertarlos en el otro (llamado **final** de la cola).

La figura 7 (“Colas y sus representaciones”) se ve en **a)** que contiene los elementos **A B y C,** A está en el frente de la cola y C en el final. En **b)** se borró un elemento de la cola. Como los elementos solo se pueden borrar desde el frente de la cola a eliminar A, B pasa a ser el nuevo frente. En **c),** cuando se insertaron los elementos D y E, fue necesario insertarlos desde el final de la cola.

Como D se insertó en la cola antes que E, será eliminada primero. El primer elemento insertado en una cola es el primero en ser eliminado. Por esta razón algunas veces las colas son denominadas listas ***fifo*** en la oposición a la pila que es una lista ***lifo***.

|  |  |  |
| --- | --- | --- |
| **A** | **B** | **C** |

**a)**

|  |  |
| --- | --- |
| **A** | **B** |

**b)**

|  |  |  |  |
| --- | --- | --- | --- |
| **A** | **B** | **C** | **D** |

**c)**

**Imagen 7 “Representación de Cola” [Elaboración propia]**

La representación de una cola como un tipo de datos abstracto es directa. *eltype* se usa para denotar el tipo de elementos de la cola y parametrizar el tipo de cola.

**abstract tydef** <<eltype>> QUEUE (eltype);

**abstract** empty(q)

QUEUE (eltype)s;

postcondition empty==(len(q)==0);

**abstract** eltype remove(q)

QUEUE (eltype) q;

precondition empty==FALSE;

**postcondition** remove == first(s’);

q==sub(q’, 1, len(s’)-1);

***abstract*** *insert (q, elt)*

QUEUE *(eltype) q;*

*eltype elt;*

***postcondition*** *q==q’+ <elt>;*

Una de las posibilidades para representar una cola el lenguaje C ex es usar un arreglo para guardar los elementos de la cola y usar dos variables para guardar las posiciones en el arreglo del último y el primer elemento de la cola dentro del arreglo. Es posible declarar una cola de enteros q mediante:

**#define** STACKSIZE 100

STRUCT queue {

**int** items [MAXQUEUE];

**int** front, rear;

**}q;**

Por supuesto que el uso de un arreglo para guardar una cola introduce la posibilidad de desborde si la cola rebasa el tamaño del arreglo punto si se hace a un lado por el momento la posibilidad de desborde y su sub desborde coma puede implantarse la operación insert (q,x) mediante las instrucciones:

Entonces la operación **insert** comprende una verificación de desborde qué ocurre cuando todo el arreglo está ocupado por elementos de la cola y se intenta insertar otro.

Tanto la pila como la cola son estructuras de datos cuyos elementos están ordenados con base en la secuencia en que se insertaron. La operación **front** recupera el último elemento insertado, en tanto que la operación toma el primer elemento que se insertó. Si hay un orden intrínseco entre los elementos, las operaciones de la pila o la cola lo ignoran, es aquí donde surge la cola de prioridad.

La cola de prioridad es una estructura de datos en la que el ordenamiento intrínseco de los elementos determina los resultados de sus operaciones básicas, hay dos tipos de colas de prioridad: la cola de prioridad ascendente y la cola de prioridad descendente.

##### -Listas

Si las listas no sólo son importantes como un medio para implementar pilas y colas, sino como estructuras de datos por derecho propio. En una lista ligada, la forma de acceso un elemento y recorriendo la lista desde el principio punto la implantación con arreglo permite acceder el enésimo elemento de un grupo por medio de una sola operación, mientras que no implantación con lista se requiere de n-operaciones. Es necesario pasar a través de los primeros n-1 elementos antes de alcanzar el elemento enésimo de una lista, ya que no hay relación entre la localidad de memoria que ocupa un elemento de una lista y su posición en la misma

ahora la implantación con lista de colas de prioridad, considerando una lista ordenada que puede usarse para representar una cola de prioridad. Para una cola de prioridad ascendente sin planta la inserción mediante la operación place, la que mantiene en orden la lista coma y la eliminación del elemento mínimo mediante la operación pop, la que elimina el primer elemento de la lista la cola te prioridad descendente puede implantarse guardando la lista en orden descendente coma en lugar de ascendente, o usando otras medidas que ya se vieron anteriormente en la parte de colas de prioridad.

# Programa

El programa se ha adjuntado junto con el ejecutable correspondiente, sin embargo se anexa aquí de igual forma junto con explicación del programa y capturas de su uso

### Renglón No.1

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

#include <malloc.h>

#include <graphics.h>

#include <time.h>

#include <dos.h>

#include <conio.h>

typedef struct nodo{

int dato;

struct nodo\* siguiente;

struct nodo\* atras;

}nodo;

nodo\* primero = NULL;

nodo\* ultimo = NULL;

void insertarNodo(int);

void buscarNodo();

void modificarNodo();

void eliminarNodo();

void desplegarListaPU();

void desplegarListaUP();

int Impresion(int);

int Impresion2(int dim);

void GranMenu();

void Dimensionamiento (int\*);

void InsersionInicial(int);

int main(){

int opcionMenu = 0;

int dim, cone=0, rer, posicion;

Dimensionamiento(&dim);

InsersionInicial(dim);

system("cls");

do{

GranMenu();

printf("\n\n Escoja una Opcion de las ya presentadas (Elija 7 si desea volver a ver el menu): ");

scanf("%d", &opcionMenu);

switch(opcionMenu){

case 1:

printf("\n\n BUSCAR UN NODO EN LA LISTA \n\n");

buscarNodo();

system("cls");

break;

case 2:

printf("\n\n MODIFICAR UN NODO DE LA LISTA \n\n");

modificarNodo();

system("cls");

break;

case 3:

printf("\n\n ELIMINAR UN NODO DE LA LISTA \n\n");

eliminarNodo();

system("cls");

break;

case 4:

printf("\n\n DESPLEGAR LISTA DE NODOS DEL PRIMERO AL UTLIMO\n\n");

Impresion(dim);

break;

case 5:

printf("\n\n DESPLEGAR LISTA DE NODOS DEL Ultimo AL Primero\n\n");

Impresion2(dim);

break;

case 6:

printf("\n\n Programa finalizado...");

break;

case 7:

printf("Lea de nuevo con atencion");

system("cls");

break;

default:

printf("\n\n Opcion No Valida \n\n");

}

}while(opcionMenu != 6);

return 0;

}

void insertarNodo(int num){

nodo\* nuevo = (nodo\*)malloc(sizeof(nodo));

nuevo->dato=num;

if(primero==NULL){

primero = nuevo;

primero->siguiente = NULL;

primero->atras = NULL;

ultimo = primero;

}else{

ultimo->siguiente = nuevo;

nuevo->siguiente = NULL;

nuevo->atras = ultimo;

ultimo = nuevo;

}

printf("\n Nodo ingresado con exito\n\n");

}

void buscarNodo(){

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

actual = primero;

int nodoBuscado = 0, encontrado = 0;

printf(" Ingrese el dato del nodo a Buscar: ");

scanf("%d", &nodoBuscado);

if(primero!=NULL){

while(actual != NULL && encontrado != 1){

if(actual->dato == nodoBuscado){

initwindow(1900,1080,"Estructura",0,0,false,true);

setlinestyle(3, 3, 3);

setcolor(BLUE);

settextstyle(BOLD\_FONT, HORIZ\_DIR, 5);

outtextxy(10,10,"El nodo con el dato si fue encontrado y se encuentra en el Cubo");

delay(4000);

closegraph();

encontrado = 1;

}

actual = actual->siguiente;

}

if(encontrado == 0){

printf("\n Nodo no Encontrado\n\n");

}

}else{

printf("\n La lista se encuentra vacia\n\n");

}

}

void modificarNodo(){

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

actual = primero;

int nodoBuscado = 0, encontrado = 0;

printf(" Ingrese el dato del nodo a Buscar para Modificar: ");

scanf("%d", &nodoBuscado);

if(primero!=NULL){

while(actual != NULL && encontrado != 1){

if(actual->dato == nodoBuscado){

printf("\n El nodo con el dato ( %d ) Encontrado", nodoBuscado);

printf("\n Ingrese el nuevo dato para este nodo: ");

scanf("%d", &actual->dato);

printf("\n Nodo modificado con exito\n\n");

encontrado = 1;

}

actual = actual->siguiente;

}

if(encontrado == 0){

printf("\n Nodo no Encontrado\n\n");

}

}else{

printf("\n La lista se encuentra vacia\n\n");

}

}

void eliminarNodo(){

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

actual = primero;

nodo\* anterior = (nodo\*)malloc(sizeof(nodo));

anterior = NULL;

int nodoBuscado = 0, encontrado = 0;

printf(" Ingrese el dato del nodo a Buscar para Eliminar: ");

scanf("%d", &nodoBuscado);

if(primero!=NULL){

while(actual != NULL && encontrado != 1){

if(actual->dato == nodoBuscado){

if(actual==primero){

primero = primero->siguiente;

primero->atras = NULL;

}else if(actual==ultimo){

anterior->siguiente = NULL;

ultimo = anterior;

}else{

anterior->siguiente = actual->siguiente;

actual->siguiente->atras = anterior;

}

initwindow(1900,1080,"Estructura",0,0,false,true);

setlinestyle(3, 3, 3);

setcolor(BLUE);

settextstyle(BOLD\_FONT, HORIZ\_DIR, 5);

outtextxy(10,10,"El nodo con el dato ya ha sido eliminado.");

delay(4000);

closegraph();

encontrado = 1;

}

anterior = actual;

actual = actual->siguiente;

}

if(encontrado == 0){

printf("\n Nodo no Encontrado\n\n");

}else{

free(anterior);

}

}else{

printf("\n La lista se encuentra vacia\n\n");

}

}

int Impresion(int dim){

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

actual=primero;

int i, colorin=1, posx=150, posy=100, cont=1, cont2=1, termx=36, termy=30, tam=3;

char bufer[dim],bufer2[dim], \*ptb, \*ptbb;

ptb=bufer;

ptbb=bufer2;

initwindow(1900,1080,"Estructura",0,0,false,true);

setlinestyle(3, 3, 3);

setcolor(2);

settextstyle(BOLD\_FONT, HORIZ\_DIR, 5);

outtextxy(10,10,"LOS DATOS QUE SE CONTIENEN EN EL CUBO SON:");

setcolor(RED);

settextstyle(BOLD\_FONT, HORIZ\_DIR, 3);

outtextxy(10,100,"Inicio");

setlinestyle(3, 3, 2);

line(90,115,140,115);

line(140,115,125,105);

line(140,115,125,125);

setcolor(1);

rectangle(posx-2,posy-1,posx+termx,posy+termy);

for(;actual!=NULL;i++,ptb++,ptbb++){

sprintf(ptb,"%d",actual->dato);

setcolor(colorin);

settextstyle(BOLD\_FONT, HORIZ\_DIR, tam);

outtextxy(posx,posy,ptb);

rectangle(posx-2,posy-1,posx+termx,posy+termy);

sprintf(ptbb,"(%d)",cont2);

setcolor(WHITE);

settextstyle(BOLD\_FONT, HORIZ\_DIR, 1);

outtextxy(posx,posy+termy,ptbb);

if(cont2==dim){

setlinestyle(3, 3, 2);

setcolor(RED);

line(posx+termx,posy+15,posx+termx+50,posy+15);

line(posx+termx+50,posy+15,posx+termx+35,posy+5);

line(posx+termx+50,posy+15,posx+termx+35,posy+25);

settextstyle(BOLD\_FONT, HORIZ\_DIR, 3);

outtextxy(posx+termx+60,posy+5,"Ultimo");

setcolor(colorin);

}

else{

setlinestyle(3, 3, 2);

setcolor(RED);

line(posx+termx,posy+15,posx+termx+40,posy+15);

line(posx+termx+40,posy+15,posx+termx+35,posy+5);

line(posx+termx+40,posy+15,posx+termx+35,posy+25);

line(posx+termx,posy+25,posx+termx+40,posy+25);

line(posx+termx,posy+25,posx+termx+5,posy+15);

line(posx+termx,posy+25,posx+termx+5,posy+35);

setcolor(colorin);

}

posx=posx+90;

if(cont==20){

posy=posy+60;

posx=150;

cont=0;

setlinestyle(3, 3, 2);

setcolor(RED);

line(90,posy+15,140,posy+15);

line(140,posy+15,125,posy+5);

line(140,posy+15,125,posy+25);

colorin++;

if(colorin>14)

colorin=1;

if(cont2==dim){

line(150,posy+15,241,posy+15);

line(170,posy+30,220,posy+30);

line(190,posy+45,200,posy+45);

}

}

cont++;

cont2++;

if(cont2>=100)

tam=1;

actual=actual->siguiente;

}

setcolor(GREEN);

settextstyle(BOLD\_FONT, HORIZ\_DIR, 4);

outtextxy(35,posy+60,"Espere un momento, la ventana se cerrara sola...");

delay(7999);

closegraph();

return 0;

}

int Impresion2(int dim){

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

actual=ultimo;

int i, colorin=1, posx=150, posy=100, cont=1, cont2=1, cont3=dim, termx=36, termy=30, tam=3;

char bufer[dim],bufer2[dim], \*ptb, \*ptbb;

ptb=bufer;

ptbb=bufer2;

initwindow(1900,1080,"Estructura",0,0,false,true);

setlinestyle(3, 3, 3);

setcolor(2);

settextstyle(BOLD\_FONT, HORIZ\_DIR, 5);

outtextxy(10,10,"LOS DATOS QUE SE CONTIENEN EN EL CUBO SON:");

setcolor(RED);

settextstyle(BOLD\_FONT, HORIZ\_DIR, 3);

outtextxy(10,100,"Ultimo");

setlinestyle(3, 3, 2);

line(90,115,140,115);

line(140,115,125,105);

line(140,115,125,125);

setcolor(1);

rectangle(posx-2,posy-1,posx+termx,posy+termy);

for(;actual!=NULL;i++,ptb++,ptbb--){

sprintf(ptb,"%d",actual->dato);

setcolor(colorin);

settextstyle(BOLD\_FONT, HORIZ\_DIR, tam);

outtextxy(posx,posy,ptb);

rectangle(posx-2,posy-1,posx+termx,posy+termy);

sprintf(ptbb,"(%d)",cont3);

setcolor(WHITE);

settextstyle(BOLD\_FONT, HORIZ\_DIR, 1);

outtextxy(posx,posy+termy,ptbb);

if(cont2==dim){

setlinestyle(3, 3, 2);

setcolor(RED);

line(posx+termx,posy+15,posx+termx+50,posy+15);

line(posx+termx+50,posy+15,posx+termx+35,posy+5);

line(posx+termx+50,posy+15,posx+termx+35,posy+25);

settextstyle(BOLD\_FONT, HORIZ\_DIR, 3);

outtextxy(posx+termx+60,posy+5,"Inicio");

setcolor(colorin);

}

else{

setlinestyle(3, 3, 2);

setcolor(RED);

line(posx+termx,posy+15,posx+termx+40,posy+15);

line(posx+termx+40,posy+15,posx+termx+35,posy+5);

line(posx+termx+40,posy+15,posx+termx+35,posy+25);

line(posx+termx,posy+25,posx+termx+40,posy+25);

line(posx+termx,posy+25,posx+termx+5,posy+15);

line(posx+termx,posy+25,posx+termx+5,posy+35);

setcolor(colorin);

}

posx=posx+90;

if(cont==20){

posy=posy+60;

posx=150;

cont=0;

setlinestyle(3, 3, 2);

setcolor(RED);

line(90,posy+15,140,posy+15);

line(140,posy+15,125,posy+5);

line(140,posy+15,125,posy+25);

colorin++;

if(colorin>14)

colorin=1;

if(cont2==dim){

line(150,posy+15,241,posy+15);

line(170,posy+30,220,posy+30);

line(190,posy+45,200,posy+45);

}

}

cont++;

cont2++;

cont3--;

if(cont2>=100)

tam=1;

actual=actual->atras;

}

setcolor(GREEN);

settextstyle(BOLD\_FONT, HORIZ\_DIR, 4);

outtextxy(35,posy+60,"Espere un momento, la ventana se cerrara sola...");

delay(7999);

closegraph();

return 0;

}

void GranMenu(){

initwindow(1900,1080,"Estructura",0,0,false,true);

setlinestyle(3, 3, 3);

setcolor(RED);

settextstyle(BOLD\_FONT, HORIZ\_DIR, 5);

outtextxy(10,10,"Menu Principal del Cubo Magico");

settextstyle(BOLD\_FONT, HORIZ\_DIR, 3);

setcolor(GREEN);

outtextxy(50,60,"1.Buscar dentro del Cubo (Por ahora primer renglon)");

outtextxy(50,100,"2.Modificar dentro del cubo (Por ahora primer renglon)");

outtextxy(50,140,"3.Eliminar algun valor del cubo (Por ahora Primer renglon)");

outtextxy(50,180,"4.Imprimir Por ahora primer renglon del principio al final");

outtextxy(50,220,"5.Imprimir por ahora primer renglon del final al principio");

outtextxy(50,260,"6.Salir del programa");

delay(4000);

closegraph();

}

void Dimensionamiento(int\*dim){

initwindow(1900,1080,"Estructura",0,0,false,true);

setlinestyle(3, 3, 3);

setcolor(RED);

settextstyle(BOLD\_FONT, HORIZ\_DIR,5);

outtextxy(10,10,"Menu Dimensionamiento del Cubo Magico");

settextstyle(BOLD\_FONT, HORIZ\_DIR, 3);

setcolor(GREEN);

outtextxy(50,60,"Este programa le va a permitir manejar los datos de un conjunto, estos se generan a su eleccion podra cambiarlos y eliminarlos");

outtextxy(50,100,"pero primero elija el tamaño de su conjunto");

delay(5000);

closegraph();

printf("Ingrese el tamaño del conjunto que va a manejar:");

scanf("%i",dim);

system("cls");

}

void InsersionInicial(int dim){

int rer=0, cone, posicion=0;

initwindow(1900,1080,"Estructura",0,0,false,true);

setlinestyle(3, 3, 3);

setcolor(RED);

settextstyle(BOLD\_FONT, HORIZ\_DIR, 5);

outtextxy(10,10,"Menu de Insersion Inicial");

settextstyle(BOLD\_FONT, HORIZ\_DIR, 3);

setcolor(GREEN);

outtextxy(50,60,"Elija un metodo de insercion que mejor le convenga");

outtextxy(50,100,"1.Numeros consecutivos");

outtextxy(50,140,"2.Numeros al Azar");

outtextxy(50,180,"3.Insercion Manual (Recomendable si desea darle un uso practico)");

delay(4000);

closegraph();

printf("Elija de acuerdo a la opcion que haya decidido:");

scanf("%d",&cone);

do{

if (cone==1){

while(rer<dim){

posicion=posicion+1;

insertarNodo(posicion);

rer++;

cone=4;

}

}

else if(cone==2){

srand (time(NULL));

while(rer<dim){

posicion=rand()%50;

insertarNodo(posicion);

rer++;

cone=4;

}

}

else if(cone==3){

system("cls");

printf("\n Inserte uno por uno y en orden los datos a introducir:\n");

while(rer<dim){

printf("\nIntroduzca el numero correspondiente a %d:",rer+1);

scanf("%d",&posicion);

insertarNodo(posicion);

rer++;

cone=4;

}

}

else

printf("\nNo eligio bien, seleccione una opcion valida...");

cone=4;

}while(cone!=4&&cone>0);

}

## Explicación del programa

El presente programa está estructurado de varias funciones, una que le permite dimensionar el tamaño del conjunto a utilizar, otro que le permite generar datos aleatorios, un menú concentrado de opciones, Funciones para establecer números al principio, al final y en cualquier lugar de en medio asignando la posición exacta. también cuenta con dos funciones adicionales una que permite eliminar algún número definido de Casillas del arreglo definidas por el usuario, y otra que, aunque tiene un ligero defecto, permiten caso de requerirse agregar Casillas asignando al mismo tiempo los valores requeridos, y por último como no debía de faltar el modo gráfico que enseña y muestra de una manera amigable para el usuario los contenidos finales del arreglo.

De nuevo podemos ver como este programa está considerado para abarcar todos los deseos básicos que el usuario podría tener durante el uso de este programa, este programa podría ser utilizado en funciones que requirieran la inserción de datos para operaciones subsecuentes, por mencionar algo. también podemos ver como el programa está diseñado para ser amigable a la vista y no confundir al usuario mediante menús enredados e necesarios, aunque en su mayoría está diseñado para un usuario obediente (o que sigue las instrucciones), también tiene medidas de seguridad para que el usuario desobediente o que no sigue las instrucciones, con el fin de reducir las probabilidades de que rompa el programa durante el uso, simplemente es un programa qué está estructurado para el uso y el manejo de gran cantidad de números, y aunque el aumento del uso de estos números podría causar dificultades en la visualización gráfica de este, no afecta de ningún modo el manejo de ellos.

# Conclusiones

El presente programa fue sin duda 1 de los programas en los que más trabajo me costó determinar cómo empezar , pues había muchos caminos que se podían tomar sin embargo estoy satisfecho con la elección que se realizó en cada 1 de los caminos con opciones a tomar, el más relevante de ellos fue el uso de las distintas funciones ocupadas en el programa aplicando así 4 de los temas más relevantes enseñados en el semestre anterior , por mencionarlos, tenemos el uso de argumentos, funciones, apuntadores, etc.

En con lo que respecta al uso de apuntadores consideró seriamente una de las herramientas más importantes a la hora de programar, pues nos permite evitarnos el cúmulo de variables basura, que lo único que hace es entorpecer muchas veces el manejo de la información albergada en ellas ,y es por ello que considero que es 1 de los recursos más valiosos para la construcción de estructuras de datos y qué será de gran ayuda en los aprendizajes y conocimientos por adquirir a lo largo de esta unidad de aprendizaje , por lo que entiendo totalmente la lógica de que sea 1 de los primeros programas a realizar en este curso punto fin.

# Referencias Bibliográficas:

Brian W. K. & Dennis M. R. (1978) Lenguaje de programación C. Pearson Education Recuperado 29 septiembre, 2020 Disponible en:<http://www.tecnica1lomas.com.ar/tutoriales/lenguaje_C.pdf>

Ricardo Ruiz (2005) Introducción Datos Estruturados. Recuperado el 28 de septiembre,2020. Disponible en: <http://programacionestructuradarrr.blogspot.com/2016/09/aritmetica-de-apuntadores.html#:~:text=Las%20variables%20de%20tipo%20apuntador,sobre%20los%20valores%20que%20almacenan.>