



Instituto Politécnico Nacional
Escuela Superior de Computo



“Reporte de los programas del 1º Parcial”

Equipo ARMY EDD

- Cervantes Martínez José Alberto
- Guerrero Pérez Diana Estefanía
- Navarro Topete José Alberto
- Rodea Azcona Erick Alberto
- Santos Méndez Ulises Jesús

Estructura de Datos

1CM1

Programa 1

“Hacer un programa para implementar el TAD fracción. “

Primero se declara la estructura fracción, la cual estará compuesta por 2 partes, una de ellas es el numerador, y la otra el denominador, ambas de tipo entero.

```
1  #include<stdio.h>
2
3  typedef struct FRACCION{
4      int numerador;
5      int denominador;
6  }fraccion;
```

Después se comenzará con la declaración y programación de las funciones. Estas serán las operaciones sumas, resta, multiplicación y división.

Primero comenzaremos con la suma. Para poder trabajar con ella necesitamos recibir dos variables de tipo fracción (en este caso a y b). Lo que se hace primero es obtener el mínimo común múltiplo de los denominadores. Una vez obtenido, se procede a sumar los numeradores como se haría de manera tradicional en las matemáticas. Finalmente se retorna el resultado por medio de una variable fracción llamada “res”.

```
10 fraccion sumar(fraccion a, fraccion b){
11     fraccion res;
12     res.denominador = min_c_m(a.denominador, b.denominador);
13     res.numerador = ((res.denominador / a.denominador) * a.numerador) + ((res.denominador / b.denominador) * b.numerador);
14     return res;
15 }
```

Para la función de resta se hace exactamente lo mismo que con la suma, solamente que como su nombre lo dice, en lugar de sumar los numeradores se van a restar

```
17 fraccion restar(fraccion a, fraccion b){
18     fraccion res;
19     res.denominador = min_c_m(a.denominador, b.denominador);
20     res.numerador = ((res.denominador / a.denominador) * a.numerador) - ((res.denominador / b.denominador) * b.numerador);
21     return res;
22 }
```

Para la función de multiplicación cambia un poco la cosa. Primero multiplicamos las multiplicaciones como se haría de manera normal, es decir, numerador por numerador y denominador por denominador. Luego se necesita obtener el máximo común divisor para poder simplificar la fracción (en caso de que sea posible), para simplificar, solamente basta con dividir el numerador y el denominador por el máximo común divisor calculado. Finalmente, el resultado se retorna por medio de una variable de tipo fracción.

```
24 ■ fraccion multiplicar(fraccion a, fraccion b){
25     fraccion res; int mcd;
26     res.numerador = a.numerador * b.numerador;
27     res.denominador = a.denominador * b.denominador;
28     mcd = max_c_d(res.numerador, res.denominador);
29     res.numerador = res.numerador / mcd;
30     res.denominador = res.denominador / mcd;
31     return res;
32 }
```

Para la fracción dividir. Primero procedemos a hacer los productos como se haría normalmente, es decir, el numerador de la fracción "a" se multiplica por el denominador de la fracción "b" y el resultado será el numerador de la fracción resultante, para el denominador se multiplica el denominador de "a" por el numerador de "b". Al igual que en la multiplicación, se necesita obtener el máximo común divisor para poder simplificar la fracción resultado en caso de ser posible.

```
34 ■ fraccion dividir(fraccion a, fraccion b){
35     fraccion res; int mcd;
36     res.numerador = a.numerador * b.denominador;
37     res.denominador = a.denominador * b.numerador;
38     mcd = max_c_d(res.numerador, res.denominador);
39     res.numerador = res.numerador / mcd;
40     res.denominador = res.denominador / mcd;
41     return res;
42 }
```

Para el máximo común divisor y el mínimo común múltiplo (max_c_d y min_c_m) se programó lo que se conoce como el "algoritmo de Euclides" (para max_c_d) y una fórmula (para el min_c_m) la cual dice que $mcm(a,b) = (a*b)/MCD(a,b)$

```

44 int max_c_d(int a, int b){
45     int mayor, menor, mcd, residuo = 1;
46     if(a > b){
47         mayor = a;
48         menor = b;
49     }else{
50         mayor = b;
51         menor = a;
52     }
53
54     while(residuo != 0){
55         residuo = mayor % menor;
56         mayor = menor;
57         menor = residuo;
58     }
59     mcd = mayor;
60     return mcd;
61 }
62
63
64 int min_c_m(int a, int b){
65     int mcm, mcd;
66     mcd = max_c_d(a,b);
67     mcm = (a * b)/mcd;
68     return mcm;
69 }

```

Ahora pasaremos al "main" que es donde se ejecutaran todas las funciones anteriormente mencionadas.

```

72 int main(){
73
74     fraccion f1,f2,res;
75     int residuo;
76
77     printf("Introduce el numerador de la fraccion\n");
78     scanf("%d",&f1.numerador);
79     printf("Introduce el denominador de la fraccion\n");
80     scanf("%d",&f1.denominador);
81     printf("%d/%d\n",f1.numerador,f1.denominador);
82     printf("Introduce el numerador de la segunda fraccion\n");
83     scanf("%d",&f2.numerador);
84     printf("Introduce el denominador de la segunda fraccion\n");
85     scanf("%d",&f2.denominador);
86     printf("%d/%d\n",f2.numerador,f2.denominador);
87
88
89     res = sumar(f1,f2);
90     printf("La suma de las fracciones es -> %d/%d\n",res.numerador,res.denominador);
91
92     res = restar(f1,f2);
93     printf("La resta de las fracciones es -> %d/%d\n",res.numerador,res.denominador);
94
95     res = multiplicar(f1,f2);
96     printf("La multiplicacion de las fracciones es -> %d/%d\n",res.numerador,res.denominador);
97
98     res = dividir(f1,f2);
99     printf("La division de las fracciones es -> %d/%d\n",res.numerador,res.denominador);
100
101     return 0;
102 }

```

RESULTADOS DE LA EJECUCIÓN:

```
Introduce el numerador de la fraccion
2
Introduce el denominador de la fraccion
3
2/3

Introduce el numerador de la segunda fraccion
4
Introduce el denominador de la segunda fraccion
5
4/5

La suma de las fracciones es -> 22/15

La resta de las fracciones es -> -2/15

La multiplicacion de las fracciones es -> 8/15

La division de las fracciones es -> 5/6

-----
Process exited after 4.243 seconds with return value 0
Presione una tecla para continuar . . . █
```

Programa 2

“Hacer un programa para implementar el TAD número complejo.”

Para la resolución de este programa, primero declaramos las funciones que vamos a ocupar

```
#include<stdio.h>
#include<math.h>
#define PI 3.1415169

//PROTOTOPOS DE FUNCION
void suma(float r[],int, float i[], int);
void resta(float e[], int, float m[], int);
void multiplicacion(float ar[], int, float ma[], int);
void division(float dr[], int, float di[], int);
void conversion(float real[],int, float imag[],int);
```

En el "Int main" declaramos nuestra variables e hicimos un menú para que cuando el usuario ingresara al programa elija que es la operación que necesita, y cuando este indique la opción, este llamara a la función que declaramos.

```

19 int main(void){
20
21     int op,i,t,t1;
22     int r=1;
23     float res;
24     float a[1],b[1];
25
26
27
28     t= sizeof(a)/sizeof(float);
29     t1=sizeof(b)/sizeof(float);
30
31     while(r==1){
32         printf("OPERACIONES CON NUMEROS COMPLEJOS\n");
33         printf("\t\t\t\t\tMENU:");
34         printf("\n1. SUMA");
35         printf("\n2. RESTA");
36         printf("\n3. MULTIPLICACION");
37         printf("\n4. DIVISION");
38         printf("\n5. CONVERSION BINOMIAL(RECTANGULAR) A POLAR");
39         printf("\n\nIngrese la operacion que desee realizar: ");
40         scanf("%d",&op);
41         for(i=0;i<=1;i++){
42             printf("\n\t\t\t\t\tIngrese el valor real %d: ",i+1);
43             scanf("%f",&a[i]);
44             printf("\n\t\t\t\t\tIngrese el valor imaginario %d: ",i+1);
45             scanf("%f",&b[i]);
46         }
47         printf("\nTus valores reales son: %.1f y %.1f, tus valores imaginarios son: %.1f y %.1f",a[0],a[1],b[0],b[1]);
48         switch(op){
49
50             case 1:
51                 suma(a,t,b,t1);
52                 break;
53
54             case 2:
55                 resta(a,t,b,t1);
56                 break;
57
58             case 3:
59                 multiplicacion(a,t,b,t1);
60                 break;

```

```

61
62         case 4:
63             division(a,t,b,t1);
64             break;
65
66         case 5:
67             conversion(a,t,b,t1);
68             break;
69
70         default:
71             printf("\nOpcion no valida");
72             break;
73     }
74
75     printf("\n\t¿Quieres realizar alguna otra operacion?\n");
76     printf("\n\t\t1.SI\t\t2.NO\n");
77     scanf("%d",&r);
78 }
79
80
81     return 0;
82 }
83

```

Esta función se llama "Suma" y como el nombre indica esta función suma los numero complejos que el usuario necesitaba sacar.

```

void suma(float r[], int t, float i[], int t1){
    float re,ima;

    re= (r[0]+r[1]);
    ima=(i[0]+i[1]);

    printf("\n\nLa suma de (%.1f+(%.1f)i)+(%.1f+(%.1f)i) es: %.1f +( %.1f)i", r[0],i[0],r[1],i[1],re,ima);
}

```

La siguiente función se llama Resta, y tal como la pasada este hace la resta de los números complejos que el usuario necesitaba sacar

```

void resta(float e[], int t, float m[], int t1){
    float re,ima;

    re= (e[0]-(e[1]));
    ima=(m[0]-(m[1]));

    printf("\n\nLa resta de (%.1f+(%.1f)i)-(%1f+(%.1f)i) es: %.1f +( %.1f)i", e[0],m[0],e[1],m[1],re,ima);
}

```

Esta Función se llama "Multiplicación" y tal como su nombre lo indica este multiplica los números complejos que el usuario ingreso.

```

void multiplicacion(float ar[], int t, float ma[], int t1){
    //i^2= -1
    float re,ima;

    re= ((ar[0]*ar[1])-(ma[0]*ma[1]));
    ima= ((ar[0]*ma[1])+(ma[0]*ar[1]));

    printf("\n\n La multiplicacion de (%.1f+(%.1f)i)(%.1f+(%.1f)i) es: %.1f + (%.1f)i", ar[0],ma[0],ar[1],ma[1],re,ima);
}

```

La Función "Division", hace la división entre los números complejos que el usuario ingreso.

```
void division(float dr[], int t, float di[], int t1){
    float re,ima,den;

    re=((dr[0]*dr[1])+(di[0]*di[1]));
    ima=((di[0]*dr[1])-(dr[0]*di[1]));
    den=pow(dr[1],2)+pow(di[1],2);

    printf("\nLa division de (%.1f+%.1f)i)/(%.1f+%.1f)i es: %f+%.1fi/%.1f",dr[0],di[0],dr[1],di[1],re,ima,den);
}
```

La siguiente Funcion se llama "Conversion", este lo que hace es que pasa de la forma que ingreso el usuario a la forma polar .

```
void conversion(float real[],int t, float imag[],int t1){
    float o,z,o1,z1,rtog;

    rtog= 180/PI;
    z= sqrt(pow(real[0],2)+pow(imag[0],2));
    o= atan2(imag[0],real[0])*rtog;
    z1= sqrt(pow(real[1],2)+pow(imag[1],2));
    o1=atan2(imag[1],real[1])*rtog;

    printf(" El polar 1 es: %.2f |%.2f",z,o);
    printf(" El polar 2 es: %.2f |%.2f",z1,o1);
}
```

Y el resultado del programa:

```
OPERACIONES CON NUMEROS COMPLEJOS
MENU:
1. SUMA
2. RESTA
3. MULTIPLICACION
4. DIVISION
5. CONVERSION BINOMIAL(RECTANGULAR) A POLAR

Ingrese la operacion que desee realizar:
```


Programa 3

“Elegir otro TAD e implementarlo.”

TAD de vectores que realizara las operaciones básicas con vectores; suma, resta, producto punto y producto cruz.

Para empezar, se crea la estructura de vector:

```
struct Vector
{
    int i = 0;
    int j = 0;
    int k = 0;
};
```

1. Función menú: En esta función se muestran una serie de opciones que son las operaciones básicas con vectores, la elección se guarda en un apuntador a entero que viene del main. Además se usa un do while para el caso en que la opción sea invalida.

```
void menu (int *opc)
{
    do
    {
        system ("cls");
        cout<<"¿Que operacion deseas realizar?"<<endl;
        cout<<"1. Suma"<<endl;
        cout<<"2. Resta"<<endl;
        cout<<"3. Producto Punto"<<endl;
        cout<<"4. Producto Cruz"<<endl;
        cin>>*opc;
    }while (*opc > 4 || *opc < 1);
}
```

2. Función Pedir_vec: Pide los valores i, j y k de los dos vectores, guardándolos en su estructura correspondiente por medio de apuntadores

```
void Pedir_vec (Vector *v1, Vector *v2)
{
    cout<<"Introduce el valor i del vector 1"<<endl;
    cin>>v1->i;

    cout<<"Introduce el valor j del vector 1"<<endl;
    cin>>v1->j;
    cout<<"Introduce el valor k del vector 1"<<endl;
    cin>>v1->k;
    system("cls");
    cout<<"Introduce el valor i del vector 2"<<endl;
    cin>>v2->i;
    cout<<"Introduce el valor j del vector 2"<<endl;
    cin>>v2->j;
    cout<<"Introduce el valor k del vector 2"<<endl;
    cin>>v2->k;
}
```

3. Función Operación: Evalúa la opción obtenida en menú por medio de un switch y dependiendo de esta realiza la operación correspondiente:

- Suma(caso 1): Suma los valores i, j y k de un vector con los correspondientes del otro vector

```
case 1:
    res.i = v1->i + v2->i;
    res.j = v1->j + v2->j;
    res.k = v1->k + v2->k;
    cout<<"El resultado de la suma es: "<<endl;
    cout<<"("<<res.i<<","<<res.j<<","<<res.k<<")"<<endl;
    break;
```

- Resta(caso 2): Resta los valores i, j y k de un vector con los correspondientes del otro vector.

```
case 2:
    res.i = v1->i - v2->i;
    res.j = v1->j - v2->j;
    res.k = v1->k - v2->k;
    cout<<"El resultado de la resta es: "<<endl;
    cout<<"("<<res.i<<","<<res.j<<","<<res.k<<")"<<endl;
    break;
```

- Producto punto(caso 3): Multiplica los valores i,j y k de un vector con los correspondientes del otro vector.

```
case 3:
    res.i = v1->i * v2->i;
    res.j = v1->j * v2->j;
    res.k = v1->k * v2->k;
    aux = res.i + res.j + res.k;
    cout<<"El resultado del producto punto es: "<<endl;
    cout<<aux;
    break;
```

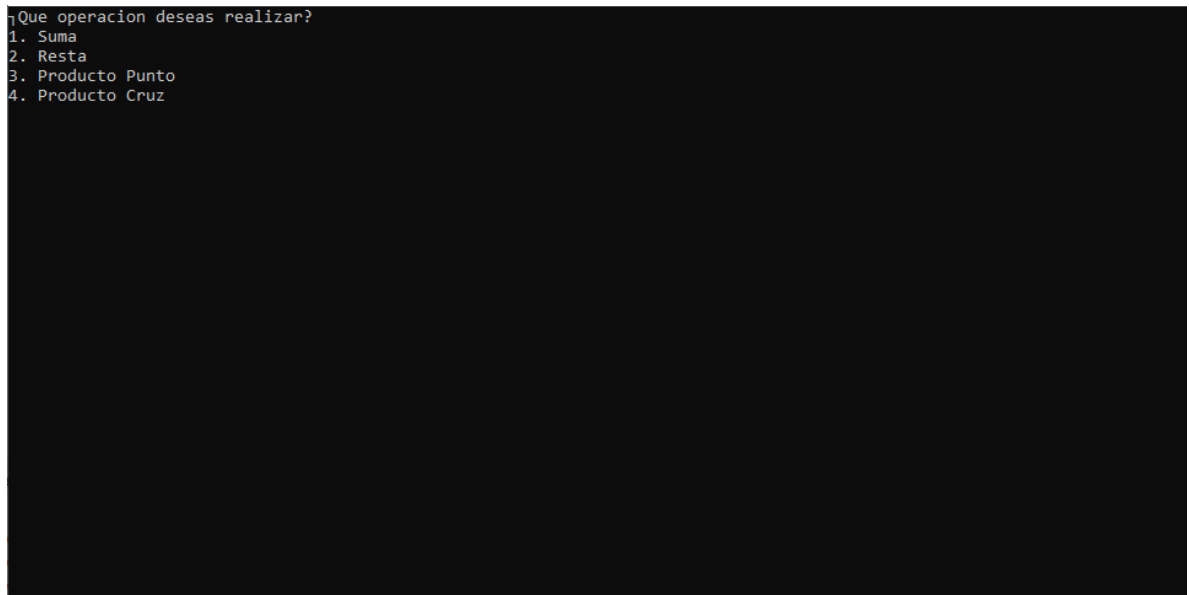
- Producto cruz(caso 4): Para el valor i del vector resultante se realiza el determinante con los valores j y k de los dos vectores, para el valor j del resultante, se realiza el determinante pero esta vez omitiendo el valor j de ambos y para el valor en k, de manera similar, pero esta vez omitiendo el valor k.

```
case 4:
    res.i = (v1->j * v2->k) - (v1->k * v2->j);
    res.j = (v1->i * v2->k) - (v1->k * v2->i);
    res.j = res.j * (-1);
    res.k = (v1->i * v2->j) - (v1->j * v1->i);
    cout<<"El resultado del producto cruz es: "<<endl;
    cout<<"("<<res.i<<","<<res.j<<","<<res.k<<")"<<endl;
    break;
```

Por ultimo se llaman las funciones en el main dentro de un do while en el caso de que se quiera realizar otra operación.

```
int main()
{
    int opc = 0;
    char r;
    Vector v1;
    Vector v2;
    do
    {
        menu(&opc);
        system("cls");
        Pedir_vec (&v1, &v2);
        system("cls");
        Operacion (&v1, &v2, &opc);
        cout<<endl;
        cout<<"¿Quieres realizar otra operacion? (s/n)"<<endl;
        cin>>r;
    }while (r == 's');
    getch();
    return 0;
}
```

Programa en funcionamiento:



```
¿Que operacion deseas realizar?
1. Suma
2. Resta
3. Producto Punto
4. Producto Cruz
```

```
El resultado del producto punto es:  
100  
¿Quieres realizar otra operacion? (s/n)
```

```
El resultado del producto cruz es:  
(19,-1,-25)  
¿Quieres realizar otra operacion? (s/n)
```

Programa 4

"Escribir un programa para hacer la implementación estática del TAD pila."

Para empezar, creamos la estructura pila:

```
struct Pila
{
    int dato[tam];
    int tope = -1;
};
```

Después creamos las funciones de la pila, junto con la función que hará de menú;

1. Función menú: Mandamos a pantalla una serie de opciones cada una de ellas numerada, y guardamos la respuesta en un apuntador que viene del main, para impedir errores, usamos un do while que reiniciara el menú si no se elige una opción inválida:

```
void menu (int *opc)
{
    do
    {
        system("cls");
        cout<<"¿Que operacion deseas realizar?:"<<endl;
        cout<<"1. Push"<<endl;
        cout<<"2. Pop"<<endl;
        cout<<"3. Mostrar pila"<<endl;
        cin>> *opc;
    }while(*opc<1 || *opc>3);
}
```

2. Función Push: Primero evalúa si el tope de la pila está en -1 (lo cual significaría que está vacía) y si no lo está, entonces pide un dato a guardar en la pila, en la posición del tope y aumenta el valor del tope en 1.

```
void Push (Pila *p)
{
    if (p->tope == tam-1)
    {
        cout<<"La pila esta llena"<<endl;
    }
    else
    {
        p->tope = p->tope + 1;
        cout<<"Introduce un dato"<<endl;
        cin>> p->dato[p->tope];
    }
}
```

3. Función Pop: Igual que la anterior, revisa si el tope esta en -1, si no lo esta imprime en pantalla el valor que esta en la posición del tope y a su vez baja el valor de este en 1.

```
void Pop (Pila *p)
{
    if (p->tope == -1)
    {
        cout<<"La pila esta vacia"<<endl;
    }
    else
    {
        system("cls");
        cout<<"Se saco el valor: "<<endl;
        cout<< p->dato[p->tope]<<endl;
        p->tope = p->tope - 1;
    }
}
```

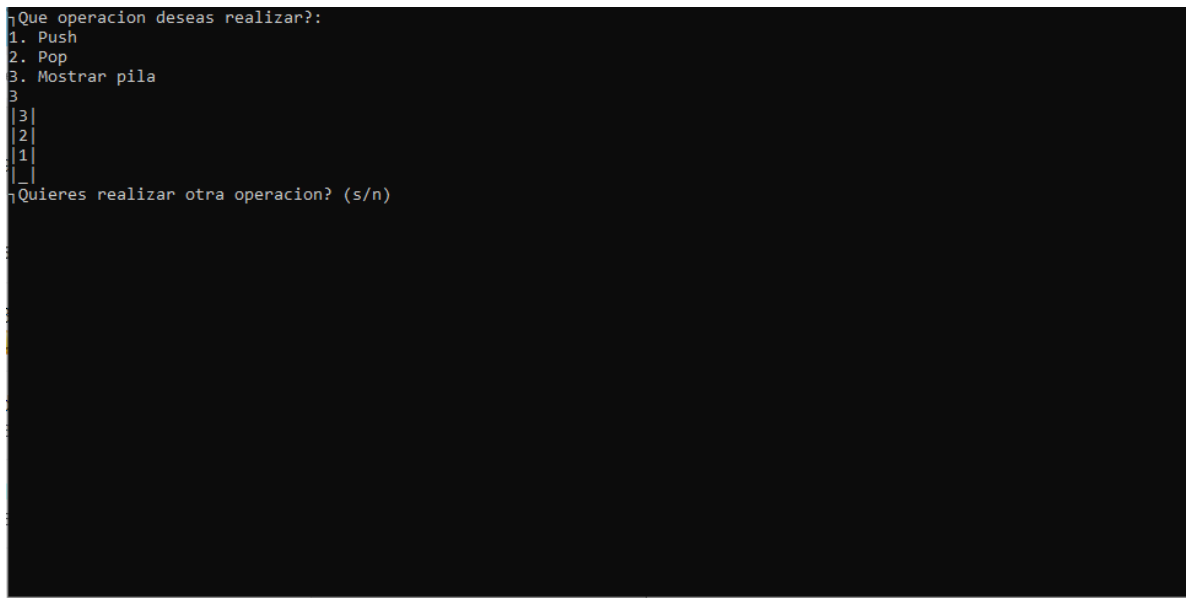
4. Función Mostrar: Esta función imprime la pila actual en pantalla, si esta no esta vacia (tope en -1), usando un for, imprime el valor entre dos separadores para darle algo de estética.

```
void Mostrar (Pila *p)
{
    int aux = 0;
    aux = p->tope;
    if (p->tope == -1)
    {
        cout<<"La pila esta vacia"<<endl;
    }
    else
    {
        for (int i = aux; i>=0; i--)
        {
            cout<<"| "<<p->dato[i]<<"| "<<endl;
        }
        cout<<"|_|"<<endl;
    }
}
```

Por último, en la función main, llamamos a las demás funciones del programa, todas ellas dentro de un do while por si se quiere repetir alguna de las operaciones.

```
int main ()
{
    Pila p1;
    int opc = 0;
    char r;
    do
    {
        system ("cls");
        menu(&opc);
        switch(opc)
        {
            case 1:
                Push(&p1);
                break;
            case 2:
                Pop(&p1);
                break;
            case 3:
                Mostrar(&p1);
                break;
        }
        cout<<"¿Quieres realizar otra operacion? (s/n)"<<endl;
        cin>>r;
    }while(r == 's');
    getch();
    return 0;
}
```

Programa en funcionamiento:



```
¿Que operacion deseas realizar?:
1. Push
2. Pop
3. Mostrar pila
3
|3|
|2|
|1|
|_|
¿Quieres realizar otra operacion? (s/n)
```

```
Se saco el valor:  
3  
¿Quieres realizar otra operacion? (s/n)
```

```
¿Que operacion deseas realizar?:  
1. Push  
2. Pop  
3. Mostrar pila  
3  
|2|  
|1|  
|_|  
¿Quieres realizar otra operacion? (s/n)
```


Programa 5

“Escribir un programa para hacer la implementación dinámica del TAD pila.”

Para la elaboración de este programa primero se definen/declaran las estructuras a utilizar, en este caso únicamente usaremos tres, las cuales son, “ELEMENTO, NODO Y PILA”

```
1  #include<stdio.h>
2  #include<stdlib.h>
3
4  //-----DECLARACION DE LAS ESTRUCTURAS-----//
5  typedef struct ELEMENTO{
6      int x;
7  }elemento;
8
9  typedef struct NODO{
10     elemento dato;
11     struct NODO *nodo_abajo;
12 }nodo;
13
14 typedef struct PILA{
15     nodo *nodo_tope;
16 }pila;
```

Como se observa en la imagen, la estructura ELEMENTO únicamente estará compuesta por una variable de tipo entero. La estructura NODO se compone de dos “partes” una de ellas es la variable de tipo elemento que se guardara en el nodo, y la otra es una dirección de memoria hacia otro nodo, en este caso, hacia el nodo de abajo. Finalmente, la estructura PILA se compone únicamente de una variable apuntador a nodo, la cual, será la encargada de darnos la dirección de memoria en donde se encuentra el nodo tope de la pila.

A continuación, se muestran las operaciones básicas de una pila:

- Iniciar la pila para su uso (iniciar_pila)
- Insertar un elemento a la pila (push_pila)
- Eliminar un elemento de la pila (pop_pila)
- Determinar si una pila está o no vacía (es_vacia)

- Determinar el elemento tope de una pila (tope_pila)
- Eliminar la pila (eliminar_pila)

Al tratarse de una PILA DINAMICA, se considera que la función de comprobar si la pila está llena o no, no es necesaria, pues en dado caso que la memoria ya no sea suficiente para poder crear los nodos de la pila, este mismo se lo hará saber al usuario. En la función de "iniciar_pila" únicamente se establece que el "nodo_tope" de la pila es "NULL" debido a que la pila estará vacía al momento de su inicialización.

```

18 //-----FUNCIONES DE LA PILA-----//
19
20 void iniciar_pila(pila *p){
21     p->nodo_tope = NULL;
22 }
23

```

En la función "push_pila" se necesita la ayuda de una variable "apuntador a nodo" auxiliar, pues en esta variable se guardará la dirección de memoria que nos generará la función "malloc". Si por alguna razón la función de malloc es incapaz de asignarnos una dirección de memoria, en la variable auxiliar se guardará un NULL, por ende, procederá a realizar la instrucción de mostrar un mensaje en pantalla de que ya no hay memoria disponible para la creación de nuevos nodos. En caso contrario, se guardará una dirección de memoria, por la cual podremos guardar el dato que queremos insertar en la pila, así como la dirección de memoria que estaba anteriormente definida como "nodo_tope". Una vez hecho lo anterior, solamente resta especificar que nuestro "nodo_tope" de la pila va a ser igual que la variable auxiliar.

```

24 void push_pila(pila *p, elemento dato){
25     nodo *aux = (nodo*)malloc(sizeof(nodo));
26     if(aux != NULL){
27         aux->dato = dato;
28         aux->nodo_abajo = p->nodo_tope;
29         p->nodo_tope = aux;
30     }else{
31         printf("ERROR. Ya no hay memoria disponible\n");
32     }
33 }

```

Para la función de "pop_pila", se creo una variable de tipo "elemento", ya que de ese tipo son las variables que almacenamos en la pila, y por lo tanto de ese tipo van a ser valores que vamos a retornar a la hora de

sacarlos de la pila. También se debe crear un "apuntador a nodo" auxiliar, este con el fin de copiar la dirección de memoria del "nodo_tope" y aplicarle un "free" una vez que se haya obtenido el dato del tope.

El funcionamiento del pop se divide en dos partes. Primero se debe verificar que la pila de la cual se desea sacar un elemento, no esté vacía. De ser ese el caso, se mandará un mensaje a pantalla en donde se especificará que la pila no tiene elementos. En caso contrario, es decir, que la pila contiene uno o más elementos, lo que se debe de hacer primero es guardar en la variable de tipo "elemento" el dato que se encuentra en el "nodo_tope" de la pila. Una vez hecho eso en variable "apuntador a nodo" aux. se guardará la dirección de memoria del "nodo_tope" actual. Ya guardada, se debe cambiar la dirección del "nodo_tope" actual por la dirección del "nodo_abajo" para que el "nodo_abajo" ahora sea el "nodo_tope" actual. Finalmente se aplica un free a la dirección de memoria guardada en aux.

```
35 elemento pop_pila(pila *p){
36     elemento element; nodo *aux;
37     if(es_vacia(p) == 1){
38         printf("ERROR. No hay elementos en la pila\n");
39     }else{
40         element = p->nodo_tope->dato;
41         aux = p->nodo_tope;
42         p->nodo_tope = p->nodo_tope->nodo_abajo;
43         free(aux);
44     }
45     return element;
46 }
```

Para la función "es_vacia" únicamente se debe verificar que la dirección de memoria de "nodo_tope" sea diferente de NULL. Al ser esto cierto, quiere decir que al menos hay un elemento en la pila.

```
47
48 int es_vacia(pila *p){
49     if(p->nodo_tope == NULL){
50         return 1;
51     }else{
52         return 0;
53     }
54 }
```

En la función "tope_pila" se debe verificar primero que la pila de la cual se quiere saber el tope, no esté vacía. De ser el caso, se mandará un mensaje a pantalla diciendo que la pila, no cuenta con un tope, debido a que está vacía. En caso contrario, por medio de la dirección de memoria del "nodo_tope" se accede al dato y se imprime.

```

56 void tope_pila(pila *p){
57     if(es_vacia(p) == 1){
58         printf("Como la pila esta vacia, no hay un tope\n");
59     }else{
60         printf("El tope de la pila es -> %d\n",p->nodo_tope->dato);
61     }
62 }

```

Por último, en la función "eliminar_pila" nuevamente se requiere el uso de un "apuntador a nodo" auxiliar, para poder hacer free a las direcciones de memoria en donde se encuentran los nodos de la pila. Este proceso se repetirá hasta que la dirección del "nodo_tope" sea igual a NULL.

```

64 void eliminar_pila(pila *p){
65     nodo *aux;
66     while(p->nodo_tope != NULL){
67         aux = p->nodo_tope;
68         p->nodo_tope = p->nodo_tope->nodo_abajo;
69         free(aux);
70     }
71 }

```

Ahora pasaremos al "main" que es donde se ejecutan cada una de las funciones programadas anteriormente.

```

73 int main(){
74     //DECLARACION DE LA PILA Y DE LOS ELEMENTOS A GUARDAR EN ELLA
75     pila p; elemento e1,e2,e3,e4,e5;
76
77     //inicializacion de las variables de tipo elemento
78     e1.x = 10; e2.x = 20; e3.x = 30; e4.x = 40; e5.x = 50;
79
80     //Se inicializa la pila para su uso
81     iniciar_pila(&p);
82
83     //Se insertan los 3 primeros elementos de la pila
84     push_pila(&p,e1);
85     push_pila(&p,e2);
86     push_pila(&p,e3);
87
88     //Se pregunta por el tope de la pila
89     tope_pila(&p);
90
91     //Se insertan los 2 valores restantes
92     push_pila(&p,e4);
93     push_pila(&p,e5);
94
95     //Se pregunta nuevamente por el tope de la pila
96     tope_pila(&p);
97
98     //Se eliminan 3 elementos de la pila
99     printf("%d\n",pop_pila(&p));
100    printf("%d\n",pop_pila(&p));
101    printf("%d\n",pop_pila(&p));
102
103 }

```

```

104 //Se pregunta nuevamente por el tope de la pila
105 tope_pila(&p);
106
107 //Se elimina la pila
108 eliminar_pila(&p);
109 |
110 //Se hace un pop (Marca error por que la pila esta vacia)
111 printf("",pop_pila(&p));
112
113
114 return 0;
115 }

```

RESULTADOS DE LA EJECUCIÓN

```

El tope de la pila es -> 30
El tope de la pila es -> 50
50
40
30
El tope de la pila es -> 20
ERROR. No hay elementos en la pila

-----
Process exited after 0.2388 seconds with return value 0
Presione una tecla para continuar . . .

```

Programa 6

“Hacer un programa para examinar los delimitadores (paréntesis, corchetes y llaves) en una expresión y determinar si es válida o no. En caso de no serlo mostrar dónde se localizó el error y en qué consiste.”

Se crean dos funciones, una que hará de menú y otra que será la que realice todo el procedimiento:

1. Función Pedir_Expresion: Pide la expresión y la guarda en el char que se mandó desde el main.

```
void Pedir_expresion (char e[30])
{
    cout<<"Ingresa la expresion:"<<endl;
    cin>>e;
}
```

2. Función Evaluar_Delimitadores: Se le manda la cadena que previamente se pidió y la evalúa, comparándola con cada uno de los delimitadores, si en alguna posición el elemento de la cadena coincide con algún delimitador, se guarda el valor de i en la posición 2 de uno de los tres arreglos: c_p (contador parentesis), c_c (contador corchetes) o c_ll (contador llaves) dependiendo de cual haya sido el delimitador coincidido, y se le aumenta en 1, el valor que está en la primera posición del arreglo.

```
void Evaluar_delimitadores (char e[30])
{
    char delimitadores[6] = {'(', ')', '{', '}', '[', ']'};
    int n = strlen(e);
    int i;
    int c_p[2];
    c_p[0] = 0;
    int c_c[2];
    c_c[0] = 0;
    int c_ll[2];
    c_ll[0] = 0;
```

```

for (i = 0; i<n; i++)
{
    switch(e[i])
    {
        case '(':
            c_p[0]++;
            c_p[1] = i+1;
            break;
        case ')':
            c_p[0]--;
            break;
        case '[':
            c_c[0]++;
            c_c[1] = i+1;
            break;
        case ']':
            c_c[0]--;
            break;
        case '{':
            c_ll[0]++;
            c_ll[1] = i+1;
            break;
        case '}':
            c_ll[0]--;
            break;
    }
}

```

Despues, se pregunta si alguno de los tres arreglos tiene operadores abiertos (valor mayor a 1), en caso de serlo con ayuda de un for se imprimen espacio en blanco por el valor que esta en la segunda posicion del arreglo y al final un indicador que en la ejecucion estara apuntando al delimitador que no se haya cerrado.

```

system("cls");
cout<<e<<endl;
if (c_p[0] == 0 && c_c[0] == 0 && c_ll[0] == 0)
{
    cout<<"La expresion no tiene errores"<<endl;
}

```

```

else
{
    if (c_p[0] > 0)
    {
        for (i = 0; i<c_p[1]-1; i++)
        {
            cout<<" ";
        }
        cout<<"^"<<endl;
        cout<<"No se cerro este parentesis";
    }
    else if(c_c[0] > 0)
    {
        for (i = 0; i<c_c[1]-1; i++)
        {
            cout<<" ";
        }
        cout<<"^"<<endl;
        cout<<"No se cerro este corchete";
    }
    else if(c_ll[0] > 0)
    {
        for (i = 0; i<c_ll[1]-1; i++)
        {
            cout<<" ";
        }
        cout<<"^"<<endl;
        cout<<"No se cerro esta llave";
    }
}
}

```

Por último llamamos desde el main a las demás funciones, todas dentro de un do while por si se quiere realizar la evaluación de otra cadena.

```

int main()
{
    char r;
    do
    {
        system("cls");
        char expresion[30];
        Pedir_expresion(expresion);
        Evaluar_delimitadores(expresion);
        cout<<endl;
        cout<<endl;
        cout<<"¿Deseas evaluar otra expresion? (s/n)"<<endl;
        cin>>r;
    }while(r == 's');
    getch();
    return 0;
}

```


Programa en funcionamiento:

```
Ingresa la expresion:  
{a+[(b+c)-d]}
```

```
{a+[(b+c)-d}  
      ^  
No se cerro este corchete  
¿Deseas evaluar otra expresion? (s/n)
```

Programa 8

“Escribir un programa para convertir una cadena interfija a posfija.”

Para realizar este programa reutilicé algo de código del TAD pila, para poder realizar las operaciones de push y pop:

```
struct Pila
{
    char dato[tam];
    int tope = -1;
    int precedencia[tam];
};
```

Push:

```
void Push (Pila *p, char aux)
{
    if (p->tope == tam-1)
    {
        cout<<"La pila esta llena"<<endl;
    }
    else
    {
        p->tope = p->tope + 1;
        p->dato[p->tope] = aux;
    }
}
```

Pop:

```
void Pop (Pila *p)
{
    if (p->tope == -1)
    {
        cout<<"La pila esta vacia"<<endl;
    }
    else
    {
        cout<< p->dato[p->tope]<<endl;
        p->tope = p->tope - 1;
    }
}
```

Cree las funciones para pedir la expresión, convertir la expresión y una para asignar valores de precedencia que ayudaran a la función a convertir.

1. Función PedirExpresion: En esta función se guarda la cadena en un char que se envia del main.

```
void PedirExpresion (char c[30])
{
    system("cls");
    cout<<"Ingresa la expresion infija"<<endl;
    cin>>c;
}
```

2. Función Conversor: En esta función con un for anidado se evalua cada uno de los elementos de la cadena con cada uno de los operadores, si en algún caso llegan a ser iguales, se llama a la función de precedencia, la cual guarda la precedencia del operador en la pila con una operación de push junto con el operador en si.

```
for (i = 0 ; i<*l ; i++)
{
    for (int j = 0 ; j<4 ; j++)
    {
        if (c[i] == operadores[j])
        {
            aux = c[i];
            i++;
            Push(&p1, aux);
            Precedencia(&p1, aux);
        }
        else if (c[i] == 'e' && c[i+1] == 'x' && c[i+2] == 'p')
        {
            aux = c[i];
            i = i+3;
            Push(&p1, aux);
            Precedencia(&p1, aux);
        }
    }
}
```

Si el elemento de la cadena no llega a concordar con ninguno de los operadores, entonces se imprime a pantalla directamente. Pero antes de eso se comprueba si la pila tiene ya mas de un elemento, de ser así, comprueba si el operador en el tope de la pila tiene menor precedencia que el que esta justo abajo, en caso afirmativo imprime el valor que esta abajo y lo quita de la pila, bajando el tope en 1.

```

        if (p1.tope >= 1)
        {
            if (p1.precedencia[p1.tope] < p1.precedencia[p1.tope-1])
            {
                if (p1.dato[p1.tope-1] == 'e')
                {
                    cout<<"exp";
                    p1.dato[p1.tope-1] = p1.dato[p1.tope];
                    p1.precedencia[p1.tope-1] = p1.precedencia[p1.tope];
                    p1.tope--;
                }
                else
                {
                    cout<<p1.dato[p1.tope-1];
                    p1.dato[p1.tope-1] = p1.dato[p1.tope];
                    p1.precedencia[p1.tope-1] = p1.precedencia[p1.tope];
                    p1.tope--;
                }
            }
        }
        cout<<c[i];
    }
}

```

Al terminar de evaluar cada uno de los elementos de la cadena, se imprimen todos los operadores que aun están en la pila con ayuda de un for.

```

for (i = p1.tope; i >= 0; i--)
{
    if (p1.dato[i] == 'e')
    {
        cout<<"exp";
    }
    else
    {
        cout<<p1.dato[i];
    }
}
cout<<endl;

```

3. Función Precedencia: Evalúa el carácter enviado desde la función Conversor y con un swithc, le asigna la precedencia directamente en la pila dependiendo del operador.

```

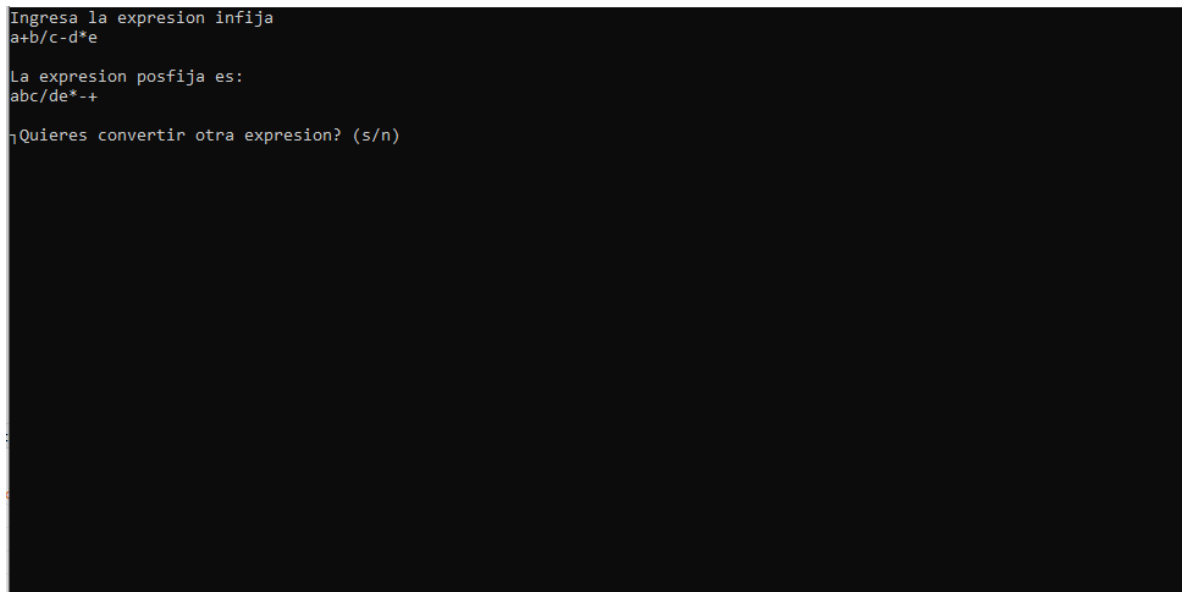
void Precedencia (Pila *p, char c)
{
    switch (c)
    {
        case '+':
            p->precedencia[p->tope] = 1;
            break;
        case '-':
            p->precedencia[p->tope] = 1;
            break;
        case '*':
            p->precedencia[p->tope] = 2;
            break;
        case '/':
            p->precedencia[p->tope] = 2;
            break;
        case 'e':
            p->precedencia[p->tope] = 3;
            break;
    }
}

```

Por último, mandamos a llamar todas las funciones en nuestro main dentro de un do while por si se quiere convertir otra expresión después.

```
int main ()
{
    char r;
    do
    {
        system("cls");
        char expresion[30];
        int lon = 0;
        PedirExpresion(expresion);
        lon = strlen (expresion);
        Conversor(expresion, &lon);
        cout<<endl;
        cout<<"¿Quieres convertir otra expresion? (s/n)"<<endl;
        cin>>r;
    }while(r == 's');
    getch();
    return 0;
}
```

Programa en funcionamiento:



```
Ingresa la expresion infija
a+b/c-d*e

La expresion posfija es:
abc/de*-+

¿Quieres convertir otra expresion? (s/n)
```

Programa 9

“Escribir un programa para hacer la implementación estática del TAD cola.”

Primero comenzamos con la definición de las estructuras a usar, las cuales son:

- ❖ ELEMENTO
- ❖ COLA

```
1  #include<stdio.h>
2  #define MAX_VALUE 5
3
4  //-----DECLARACION DE LAS ESTRUCTURAS-----//
5  typedef struct ELEMENTO{
6      int x;
7  }elemento;
8
9  typedef struct COLA{
10     elemento A[MAX_VALUE];
11     int frente, final;
12     int num_elemento;
13 }cola;
```

En la estructura “ELEMENTO” en este caso, únicamente estará formada por una variable de tipo entero, el cual va a ser almacenado en la cola.

En el caso de la estructura “COLA” se compondrá de 4 partes, una de ellas es el arreglo en donde almacenaremos las variables de tipo “elemento”, las otras 3, serán variables de tipo entero, estas son “frente”(para saber cuál es el elemento que se encuentra primero en la cola), “final” (para saber que elemento se encuentra al final de la cola) y “num_elemento” (para saber el total de elementos que hay en la cola).

Ahora procederemos a mostrar las operaciones básicas de una cola, las cuales son:

- Iniciar una cola para su uso (iniciar_cola)
- Insertar un elemento a la cola (encolar)
- Eliminar un elemento a la cola (desencolar)
- Saber si la cola está vacía (es_vacia)

- Saber si la cola está llena (es_llena)
- Obtener el elemento que se encuentra al frente de la cola (frente)
- Obtener el elemento que se encuentra al final de la cola (final)
- Obtener el elemento que se encuentra en una posición "n" de la cola (element)
- Obtener el tamaño de la cola (tam)
- Eliminar la cola

Para la función "iniciar_cola" de forma estática, se debe iniciar el "frente" y "final" con valor de -1 (ya que como sabemos, la primera posición de un arreglo suele tomarse generalmente como 0, y al iniciar la cola, aun no tenemos un elemento guardado), y "num_elemento" con valor de 0.

```
17 void iniciar_cola(cola *c){
18     c->frente = -1; c->final = -1; c->num_elemento = 0;
19 }
```

Para la función "encolar" se toman en cuenta varios casos. El primero cuando la cola está llena. Si la cola está llena, quiere decir que es imposible meter un dato más, pues no hay espacio en donde hacerlo.

El segundo es cuando la cola está vacía y se va a insertar el primer elemento. Este caso se considera especial, ya que, al tratarse del primer y único elemento de la cola, también se convierte en el último, por lo tanto, en este caso hay que recorrer tanto el frente como el final de cola. Una vez hecho eso, se debe guardar el dato en una posición del arreglo, dicha posición estará dado por la variable "final", y finalmente se debe incrementar en 1 el "num_elemento".

Otro caso especial es cuando ya existen elementos en la cola, pero quedan espacios libres al comienzo del arreglo. Para aprovechar esos espacios lo que se hizo fue poner la siguiente condición; "Si el frente de tu cola, no es la posición 0 del arreglo Y el final de tu cola es igual a MAX_VALUE (valor máximo de datos permitidos para almacenar)-1".

La parte de "Si el frente de tu cola, no es la posición 0 del arreglo" quiere decir que al menos la posición 0 esta vacía, por lo cual podemos almacenar un dato ahí. La parte de "el final de tu cola es igual a MAX_VALUE (valor máximo de datos permitidos para almacenar)-1" se refiere a que ya llegamos a la última posición del arreglo. Si ambas condiciones se cumplen, entonces el valor del "final" de la cola se establece en 0, y el dato se guarda en la posición dada por "final". Ya por ultimo se incrementa el "num_elemento" en 1.

En caso de que una de las dos condiciones no se cumpla, simplemente se sigue recorriendo el "final" de la cola, almacenando el dato en la posición del arreglo que indique el "final" e incrementando el "num_elemento" en 1.

```
21 void encolar(coola *c, elemento dato){
22     if(es_vacia(c) == 1){
23         c->frente++; c->final++;
24         c->A[c->final] = dato;
25         c->num_elemento++;
26     }else{
27         if(es_llena(c) == 1){
28             printf("Error. La cola esta llena.\n");
29         }else{
30             if(c->frente != 0 && c->final == MAX_VALUE-1){
31                 c->final = 0;
32                 c->A[c->final] = dato;
33                 c->num_elemento++;
34             }else{
35                 c->final++;
36                 c->A[c->final] = dato;
37                 c->num_elemento++;
38             }
39         }
40     }
41 }
```

Para la función "desencolar" se necesita una variable de tipo elemento para poder almacenar el dato que se va a desencolar. Primero debemos preguntar si la cola está vacía, de ser éste el caso, se mandará un mensaje a pantalla diciendo que no hay elementos para desencolar. En caso contrario, se tiene que almacenar en la variable de tipo elemento, el dato que se encuentra en la posición "frente" de la cola, una vez hecho eso se debe incrementar el "frente" y decrementar el "num_elementos".


```

43 elemento desencolar(cola *c){
44     elemento element;
45     if(es_vacia(c) == 1){
46         printf("ERROR. No hay elementos para desencolar.\n");
47     }else{
48         element = c->A[c->frente];
49         c->frente++;
50         c->num_elemento--;
51         return element;
52     }
53 }

```

Para las funciones "es_vacia" y "es_llena" solamente basta con utilizar la variable "num_elementos". Si está variable es 0, entonces quiere decir que la cola está vacía. En caso contrario, si el valor de la variable es igual al número máximo de elementos permitidos para guardar en el arreglo (MAX_VALUE) quiere decir que la pila está llena.

```

54 int es_vacia(cola *c){
55     if(c->num_elemento == 0){
56         return 1;
57     }else{
58         return 0;
59     }
60 }
61 int es_llena(cola *c){
62     if(c->num_elemento == MAX_VALUE){
63         return 1;
64     }else{
65         return 0;
66     }
67 }

```

Para la función "frente" y "final" únicamente se debe de imprimir el dato que se encuentra en la posición dada por el valor de "frente" y "final" según sea el caso.

```

68 void frente(cola *c){
69     printf("El frente de la cola es -> %d\n", c->A[c->frente]);
70 }
71 void final(cola *c){
72     printf("El final de la cola es -> %d\n", c->A[c->final]);
73 }

```

Para la función "element", la cual te muestra el dato que se encuentra en una posición "n" de cola, se utilizan dos variables, un contador y una bandera, la cual tomara el valor que

tenga "frente" en ese momento.

Primero se debemos asegurar que la posición "n" se encuentre entre uno de los valores validos de nuestro arreglo, es decir, que no sea un número mayor al número máximo de elementos permitidos, de no cumplirse esta condición, se mandará un mensaje de error diciendo que se esta tratando de acceder a una posición inexistente de la cola. En caso contrario por medio de un bucle, se comienzan a utilizar las variables previamente mencionadas (contador y bandera), mientras que el contador sea MENOR que la posición "n" tanto la bandera como el contador se van a ir incrementando. Ahora dentro del mismo bucle hay una condición, la cual es que si la bandera llega a la última posición del arreglo de la cola, la bandera se pondrá con un valor de -1 (este caso aplica para cuando el elemento buscado se encuentra en alguna de las posiciones iniciales del arreglo). Una vez que se termine el bucle, es decir, que el contador sea igual que la posición "n" entonces se imprimirá el dato del arreglo que está en la posición "bandera" de la cola.

```
74 void element(cola *c, int posicion){
75
76     int cont = 1, bandera = c->frente;
77
78     if(posicion > MAX_VALUE){
79         printf("ERROR. Se esta tratando de acceder a una posicion inexistente\n");
80     }else{
81         while(cont < posicion){
82             if(bandera == MAX_VALUE-1){
83                 bandera = -1;
84             }
85             bandera++; cont++;
86         }
87         printf("El dato que se encuentra en la posicion %d es -> %d\n", posicion, c->A[bandera]);
88     }
89 }
```

Para la función "eliminar_cola" basta con simplemente llamar a la función "iniciar_cola", pues se trata de una cola estática.

```
90 void eliminar_cola(cola *c){
91     iniciar_cola(c);
92 }
```

Para la función "tam" únicamente se debe de retornar el valor de "num_elemento" de la cola.

```

93 int tam(cola *c){
94     return c->num_elemento;
95 }

```

Ahora vamos con el "main" que es donde se ejecutaran las funciones anteriormente mostradas:

```

98 int main(){
99
100     cola c; elemento e1,e2,e3,e4,e5,e6;
101     e1.x = 10; e2.x = 20; e3.x = 30; e4.x = 40; e5.x = 50; e6.x = 60;
102     //elemento *p_A;
103     //p_A = &c.A;
104
105     //printf("%p\n",p_A);
106
107     //Se llama a iniciarCola
108     iniciarCola(&c);
109
110     //Se insertan los 3 primeros elementos
111     encolar(&c,e1);
112     encolar(&c,e2);
113     encolar(&c,e3);
114
115     //Se pregunta por el frente y final de la cola
116     frente(&c);
117     final(&c);
118
119     //Se sacan 2 elementos de la cola
120     printf("El elemento desencolado es -> %d\n", desencolar(&c));
121     printf("El elemento desencolado es -> %d\n", desencolar(&c));
122

```

```

123     //Se pregunta por el frente, final y tamaño de la cola
124     frente(&c);
125     final(&c);
126     printf("El tam de la cola es -> %d\n",tam(&c));
127
128     //Se meten 3 elementos mas
129     encolar(&c,e4);
130     encolar(&c,e5);
131     encolar(&c,e6);
132
133     //Se pregunta por el frente, final y tamaño de la cola
134     frente(&c);
135     final(&c);
136     printf("El tam de la cola es -> %d\n",tam(&c));
137
138     //Se encola otro elemento
139     encolar(&c,e1);
140
141     //Se pregunta por el frente, final y tamaño de la cola
142     frente(&c);
143     final(&c);
144     printf("El tam de la cola es -> %d\n",tam(&c));
145
146     //Se trata de encolar otro elemento (MARCA EL ERROR, PUES LA COLA YA ESTA LLENA)
147     encolar(&c,e2);

```

```

149 //Se busca el dato en la posicion 5 de la cola
150 element(&c,5);
151
152 //Se saca un elemento de la cola
153 desencolar(&c);
154 //se pregunta por el frente, final y tam de la cola
155 frente(&c);
156 final(&c);
157 printf("El tam de la cola es -> %d\n",tam(&c));
158
159 //Se elimina la cola
160 eliminarCola(&c);
161
162 //Se trata de desencolar un elemento (MARCA ERROR, PUES LA COLA ESTA VACIA)
163 printf("El elemento desencolado es -> %d", desencolar(&c));
164
165 return 0;
166 }

```

RESULTADOS DE LA EJECUCIÓN

```

El frente de la cola es -> 10
El final de la cola es -> 30
El elemento desencolado es -> 10
El elemento desencolado es -> 20
El frente de la cola es -> 30
El final de la cola es -> 30
El tam de la cola es -> 1
El frente de la cola es -> 30
El final de la cola es -> 60
El tam de la cola es -> 4
El frente de la cola es -> 30
El final de la cola es -> 10
El tam de la cola es -> 5
Error. La cola esta llena.
El dato que se encuentra en la posicion 5 es -> 10
El frente de la cola es -> 40
El final de la cola es -> 10
El tam de la cola es -> 4
ERROR. No hay elementos para desencolar.

```

Programa 10

“Escribir un programa para hacer la implementación dinámica del TAD cola.”

Primero comenzamos con la definición de las estructuras a usar, las cuales son:

- ✓ ELEMENTO
- ✓ NODO
- ✓ COLA

```
1  #include<stdio.h>
2  #include<stdlib.h>
3
4  //Definicion de las estructuras
5
6  typedef struct ELEMENTO{
7      int x;
8  }elemento;
9
10 typedef struct NODO{
11     elemento dato;
12     struct NODO *nodo_siguiente;
13 }nodo;
14
15 typedef struct COLA{
16     nodo *frente;
17     nodo *final;
18     int num_elementos;
19 }cola;
```

La estructura “ELEMENTO” únicamente estará compuesta por una variable de tipo entero.

La estructura NODO se compone de dos “partes” una de ellas es la variable de tipo elemento que se guardara en el nodo, y la otra es una dirección de memoria hacia otro nodo, en este caso, hacia el “nodo_siguiente”. Finalmente, la estructura COLA se compone de 3 partes, dos apuntadores a nodo los cuales son el nodo del “frente” de la cola, y el nodo “final” de la cola, y la ultima parte es el “num_elementos” de la cola.

Las operaciones en una cola dinámica son:

- Iniciar una cola para su uso (iniciar_cola)
- Insertar un elemento a la cola (encolar)
- Eliminar un elemento a la cola (desencolar)
- Saber si la cola está vacía (es_vacia)
- Obtener el elemento que se encuentra al frente de la cola (frente)
- Obtener el elemento que se encuentra al final de la cola (final)
- Obtener el elemento que se encuentra en una posición "n" de la cola (element)
- Obtener el tamaño de la cola (tam)
- Eliminar la cola

Para la función "iniciar_cola" se debe poner los apuntadores a nodo de la cola (frente y final) con valor NULL, y el "num_elementos" con valor 0.

```
22
23 void iniciar_cola(cola *c){
24     c->frente = NULL; c->final = NULL; c->num_elementos = 0;
25 }
26
```

Para la función "encolar" se necesita la ayuda de una variable "apuntador a nodo", el cual se identificará como "nuevo_nodo". Si por alguna razón malloc no nos puede asignar un espacio de la memoria, entonces se mandará un mensaje a la pantalla de error indicando que ya no hay memoria disponible. En caso de que, si se nos asigne una dirección de memoria, entonces se procede a preguntar si la cola está vacía. Al igual que con la cola estática, este es un caso especial, pues al no haber ningún elemento en la lista, el primero en ingresar, también se va a convertir en el último. Si ese es el caso, entonces el apuntador a nodo "frente" y también "final" se les asigna la dirección de memoria que guarda "nuevo_nodo", después ya con la dirección de memoria de "final" podemos guardar el dato y también la dirección de memoria de "nodo_siguiente", la cual va a ser NULL (pues como se dijo anteriormente, es el único elemento en la cola, por lo tanto no apunta a algún otro nodo). Finalmente incrementamos "num_elementos" en 1.

En caso de que ya existan elementos en la cola, solamente se procede a asignarle a "nodo_siguiente" la dirección de memoria de "nuevo_nodo" usando aun la dirección actual de "final". Una vez hecho eso, hay que asignarle a "final" la dirección de memoria de "nuevo_nodo" y ya con esa dirección de memoria, proceder a guardar el dato y la dirección de "nodo_siguiente", la cual será NULL (esto porque al ser el último nodo, no se

encuentra uno atrás de él), y finalmente incrementar "num_elementos" en 1.

```
27 void encolar(coola *c, elemento dato){
28
29     nodo *nuevo_nodo = (nodo*)malloc(sizeof(nodo));
30
31     if(nuevo_nodo == NULL){
32         printf("ERROR. Ya no hay memoria disponible\n");
33     }
34
35     if(es_vacia(c) == 1){
36         c->frente = nuevo_nodo; c->final = nuevo_nodo;
37         c->final->dato = dato; c->final->nodo_siguiente = NULL;
38         c->num_elementos++;
39     }else{
40         c->final->nodo_siguiente = nuevo_nodo;
41         c->final = nuevo_nodo;
42         c->final->dato = dato;
43         c->final->nodo_siguiente = NULL;
44         c->num_elementos++;
45     }
46 }
47
```

Para "desencolar" se necesita un apuntador a nodo auxiliar, en el cual se guardará la dirección de memoria a la que se le aplicará un "free". Pero antes de eso, primero se debe preguntar si la cola esta vacía, de ser eso cierto, se mandara un mensaje de error. En caso contrario lo que se hace es asignarle a la variable auxiliar la dirección de memoria que tiene "frente", una vez hecho eso, guardamos el dato que vamos a retornar en una variable de tipo "elemento", después se procede a mover el "frente" de la cola, esto se logra haciendo que el "frente" guarde la dirección de memoria de su "nodo_siguiente", se le aplica un free a la variable auxiliar y finalmente se decrementa en 1 a "num_elementos".

```
48 elemento desencolar(coola *c){
49
50     nodo *aux; elemento element;
51
52     if(es_vacia(c) == 1){
53         printf("ERROR. Subdesbordamiento de cola\n");
54     }else{
55         aux = c->frente;
56         element = aux->dato;
57         c->frente = c->frente->nodo_siguiente;
58         free(aux);
59         c->num_elementos--;
60         return element;
61     }
62
63 }
```

Para la función “es_vacia” únicamente se debe de verificar que el apuntador a nodo “final” sea NULL, en caso de no serlo, quiere decir que al menos hay un elemento en la cola.

```
65 int es_vacia(cola *c){
66     if(c->final == NULL){
67         return 1;
68     }else{
69         return 0;
70     }
71 }
72
```

Para la función “frente” y “final” basta simplemente con acceder al dato por medio de las direcciones de memoria que contienen los apuntadores a nodo “frente” y “final”.

Claramente, antes de eso, primero se debe verificar que la cola no este vacía, de ser el caso, se mandará un mensaje de error.

```
73 void frente(cola *c){
74     if(es_vacia(c) == 1){
75         printf("ERROR. La cola no tiene un frente debido a que esta vacia\n");
76     }else{
77         printf("El frente de la cola es -> %d\n", c->frente->dato);
78     }
79 }
80
81 void final(cola *c){
82     if(es_vacia(c) == 1){
83         printf("ERROR. La cola no tiene un final debido a que esta vacia\n");
84     }else{
85         printf("El final de la cola es -> %d\n", c->final->dato);
86     }
87 }
88
```

Para la función de “element” se utilizará un contador y una variable apuntador a nodo “auxiliar”, a esta variable se le asignará la dirección de memoria de “frente”. Primero se pregunta si la cola está vacía, de ser el caso, se mandará un mensaje de error. En caso contrario ahora se pregunta si la posición “n” es válida para los elementos actuales en la cola, si no es así, se mandará un mensaje de error. En caso contrario ahora si se procederá con el uso de las variables antes mencionadas. Al igual que como se hizo con la cola estática, se usará un bucle, el consistirá en hacer que el contador sea igual a la posición “n”, mientras esta condición no se cumpla, lo que se hará es ir cambiando la dirección de memoria de “auxiliar” a la dirección de memoria que tiene el “nodo_siguiente” e incrementar el contador. Cuando el contador sea igual que la posición “n”, entonces el bucle se termina y se imprime el dato que se encuentra en

la dirección de memoria que haya quedado guardada en "aux" en ese momento.

```
89 void element(cola *c, int posicion){
90     int cont = 1; nodo *aux = c->frente;
91
92     if(es_vacia(c) == 1){
93         printf("La cola esta vacia, no hay elementos para buscar\n");
94     }else{
95         if(posicion <= c->num_elementos){
96             while(cont < posicion){
97                 aux = aux->nodo_siguiete;
98                 cont++;
99             }
100             printf("El elemento en la posicion %d es -> %d\n", posicion, aux->dato);
101         }else{
102             printf("ERROR. Esta tratando de acceder a una posicion inexistente\n");
103         }
104     }
105 }
```

Al igual que en la cola estatica, para la función "tam", únicamente basta con retornar el valor de "num_elementos".

```
106 int tam(cola *c){
107     return c->num_elementos;
108 }
```

Para la función "eliminar" también se utilizará un apuntador a nodo "auxiliar". En esta variable se guardarán las direcciones de memoria a las que se les deberá hacer "free". Este proceso se repetirá hasta que el apuntador a nodo "frente" sea NULL.

Al ser el frente NULL por ende el "final" también será NULL, y como la pila fue eliminada, entonces el "num_elementos" es 0.

```
110 void eliminar_cola(cola *c){
111     nodo *aux;
112     while(c->frente != NULL){
113         aux = c->frente;
114         c->frente = c->frente->nodo_siguiete;
115         free(aux);
116     }
117     c->num_elementos = 0;
118     c->final = NULL;
119 }
```

Ahora vamos con el "main" que es donde se ejecutaran las funciones previamente mostradas:

```

123 int main(){
124     cola c; elemento e1,e2,e3,e4,e5,e6;
125     e1.x = 10; e2.x = 20; e3.x = 30; e4.x = 40; e5.x = 50; e6.x = 60;
126
127     //elemento *p_A;
128     //p_A = &c.A;
129
130     //printf("%p\n",p_A);
131
132     //Se llama a iniciarCola
133     iniciarCola(&c);
134
135     //Se insertan los 3 primeros elementos
136     encolar(&c,e1);
137     encolar(&c,e2);
138     encolar(&c,e3);
139
140     //Se pregunta por el frente y final de la cola
141     frente(&c);
142     final(&c);
143
144     //Se sacan 2 elementos de la cola
145     printf("El elemento desencolado es -> %d\n", desencolar(&c));
146     printf("El elemento desencolado es -> %d\n", desencolar(&c));

```

```

148     //Se pregunta por el frente, final y tamaño de la cola
149     frente(&c);
150     final(&c);
151     printf("El tam de la cola es -> %d\n",tam(&c));
152
153     //Se meten 3 elementos mas
154     encolar(&c,e4);
155     encolar(&c,e5);
156     encolar(&c,e6);
157
158     //Se pregunta por el frente, final y tamaño de la cola
159     frente(&c);
160     final(&c);
161     printf("El tam de la cola es -> %d\n",tam(&c));
162
163     //Se encola otro elemento
164     encolar(&c,e1);
165
166     //Se pregunta por el frente, final y tamaño de la cola
167     frente(&c);
168     final(&c);
169     printf("El tam de la cola es -> %d\n",tam(&c));
170
171     //----Se trata de encolar otro elemento
172     encolar(&c,e2);

```

```

176 //----Se pregunta por el frente, final y tamaño de la cola
177 frente(&c);
178 final(&c);
179 printf("El tam de la cola es -> %d\n",tam(&c));
180
181 //Se busca el dato en la posicion 5 de la cola
182 element(&c,7);
183
184 //Se saca un elemento de la cola
185 desencolar(&c);
186 //se pregunta por el frente, final y tam de la cola
187 frente(&c);
188 final(&c);
189 printf("El tam de la cola es -> %d\n",tam(&c));
190
191
192
193 //Se elimina la cola
194 eliminarCola(&c);
195
196 //se pregunta por el frente, final y tam de la cola
197 frente(&c);
198 final(&c);
199 printf("El tam de la cola es -> %d\n",tam(&c));
200
201 return 0;

```

RESULTADOS DE LA EJECUCIÓN:

```

El frente de la cola es -> 10
El final de la cola es -> 30
El elemento desencolado es -> 10
El elemento desencolado es -> 20
El frente de la cola es -> 30
El final de la cola es -> 30
El tam de la cola es -> 1
El frente de la cola es -> 30
El final de la cola es -> 60
El tam de la cola es -> 4
El frente de la cola es -> 30
El final de la cola es -> 10
El tam de la cola es -> 5
El frente de la cola es -> 30
El final de la cola es -> 20
El tam de la cola es -> 6
El elemento en la posicion 5 es -> 10
El frente de la cola es -> 40
El final de la cola es -> 20
El tam de la cola es -> 5
ERROR. La cola no tiene un frente debido a que esta vacia
ERROR. La cola no tiene un final debido a que esta vacia
El tam de la cola es -> 0

```

Programa 14

“Escribir un programa para hacer una implementación estática del TAD lista lineal.”

Para elaborar este programa primero declaramos la estructura que vamos a ocupar

```
8  typedef int TipoDato;
9
10 typedef struct alm{
11     int numelm;
12     TipoDato dato[MAX];
13 }Nodo;
```

En la int main se hizo lo que es un menú para que el usuario pueda usar la lista, las declaraciones de las variables y apuntadores.

```
39 int main (void){
40     Nodo LU, *Lista;
41     Lista=&LU;
42     CrearOVaciarLista(Lista);
43     int posicion, opcion, nlista, nelista, ubicacion;
44     TipoDato elemento, eliminado;
45
46     do{
47         system("cls");
48         printf("\nBienvenido al programa de dos listas\n");
49         printf("Insertar un elemento al inicio de la lista.....[1]\n");
50         printf("Insertar un elemento al final de la lista.....[2]\n");
51         printf("Insertar un elemento en un punto intermedio dentro del dominio de la lista...[3]\n");
52         printf("Eliminar un elemento al inicio de la lista.....[4]\n");
53         printf("Eliminar un elemento al final de la lista.....[5]\n");
54         printf("Eliminar un elemento en un punto intermedio dentro del dominio de la lista...[6]\n");
55         printf("Vaciar lista.....[7]\n");
56         printf("Salir del programa.....[8]\n");
57
58         printf("\nPrimer lista:\n");
59         ImpresionLista(Lista);
60         //ImprimirNelementos(Lista);
61
62         printf("\nEl valor eliminado de la primera lista fue: %d\n", eliminado);
63
64         printf("\nPor favor escoja una opcion del menu:\n");
65         scanf("%d",&opcion);
66
67         if(opcion==1){
68             printf("Introduzca el dato de su preferencia\n");
69             scanf("%d",&elemento);
70             InsertarInicio(Lista,elemento);
71         }
```

```

72     else if(opcion==2){
73         printf("Introduzca el dato de su preferencia\n");
74         scanf("%d",&elemento);
75         InsertarFinal(Lista,elemento);
76     }
77     else if(opcion==3){
78         printf("Introduzca el dato de su preferencia\n");
79         scanf("%d",&elemento);
80         printf("Introduzca la posicion de su preferencia\n");
81         scanf("%d",&posicion);
82         InsertarCPI(Lista,elemento,posicion);
83     }
84     else if(opcion==4){
85         eliminado=EliminarInicio(Lista);
86         nlista=nlista;
87     }
88     else if(opcion==5){
89         eliminado=EliminarFinal(Lista);
90         nlista=nlista;
91     }
92     else if(opcion==6){
93         printf("Introduzca la posicion de su preferencia\n");
94         scanf("%d",&posicion);
95         eliminado=EliminarCPI(Lista,posicion);
96         nlista=nlista;
97     }
98     else if(opcion==7){
99         CrearOVaciarLista(Lista);
100    }
101    else if(opcion==8){
102        printf("Nos vemos\n");
103        printf("\n");
104    }
105    else
106        printf("Opcion invalida\n");
107        printf("\n");
108    }while(opcion!=11);
109
110    return 0;
111 }
112

```

Después declaramos una función de "Crear o Vaciar listas" y esta como nos indican en el nombre nos va a crear una lista desde 0 o bien si ya había una se va a vaciar

```
void CrearOVaciarLista(Nodo* lista){
    int cont=0;
    do{
        lista->dato[cont]=COMPV;
        cont++;
    }while(cont!=MAX);
    lista->numelm=0;
    return;
}
```

Otra función que ocupamos se llama "Impresión Lista", esta nos indica al inicio de la ejecución si nuestra lista esta vacía, si no esta vacía este nos dará una impresión de la lista que está en la primera posición, pero si no cumple con la condición de que sea diferente al numero de la lista que esta apuntando, se le imprimirá el número de elementos de la lista.

```
122
123 void ImpresionLista(Nodo* lista){
124     int cont=0;
125     if(lista->numelm==0){
126         printf("Tu lista esta vacia\n");
127         return;
128     }
129     else
130     do{
131         printf("%d ",lista->dato[cont]);
132         cont++;
133     }while(cont!=lista->numelm);
134     printf("\nNumero de elementos en tu lista: %d\n",lista->numelm);
135
136     return;
137 }
138
```

Esta función se llama "Imprimir Número de Elementos" y solo sirve para la impresión de números de elementos de la lista.

```
139 void ImprimirNelementos(Nodo *lista){
140     printf("\nNumero de elementos en tu lista: %d\n",lista->numelm);
141     return;
142 }
```

Entonces, la función de a continuación se llama "Insertar Final" esto lo que va a ser es insertar un elemento a la lista pero hasta al final de la declaración de la lista, pero primero comprobamos si esta llena, si es así se le avisa al usuario que esta llena, y regresara al menú, pero si no

entonces podrá poner el elemento que desea poner el usuario al final de la lista

```
143
144 void InsertarFinal(Nodo* lista, TipoDato elemento){
145     if(lista->numelm==MAX){
146         printf("Tu lista esta llena\n");
147         return;
148     }
149     else
150     lista->dato[lista->numelm]=elemento;
151     lista->numelm++;
152     return;
153 }
154
```

Esta función es parecido al que acabamos de comentar pero la diferencia es que este podrá el elemento que desea el usuario hasta el inicio de la lista, este se llama "Insertar Inicio" y hace un poco lo mismo como en Insertar Final pero a este hace verifica si en la posición que esta en la lista esta llena, si no es así, este verifica si la lista en la posición que esta este es menor que 1 lo que hará es llamar a la función Insertar Final, si no es así pondrá el elemento en la primera posición.

```
void InsertarInicio(Nodo* lista, TipoDato elemento){
    if(lista->numelm==MAX){
        printf("Tu lista esta llena\n");
        return;
    }
    else if(lista->numelm<1){
        InsertarFinal(lista,elemento);
        return;
    }
    int aux,apy;
    aux=lista->numelm;
    aux--;
    apy=lista->numelm;
    do{
        lista->dato[apy]=lista->dato[aux];
        apy--;
        aux--;
    }while(apy!=0);
    lista->dato[0]=elemento;
    lista->numelm++;
    return;
}
```

Esta función se llama Eliminar Final, esta sirve para poder eliminar el ultimo elemento que tienes en la lista, pero antes que haga eso verifica si esta vacía, si es así, le avisa al usuario que esta vacía la lista, si no este borra el ultimo elemento de la lista.

```
177
178  TipoDato EliminarFinal(Nodo* lista){
179      if(lista->numelm==0){
180          printf("Tu lista esta vacia\n");
181      }
182      int aux;
183      TipoDato rpd;
184      aux=lista->numelm;
185      aux--;
186      rpd=lista->dato[aux];
187      lista->dato[aux]=COMPV;
188      lista->numelm--;
189      return rpd;
190  }
```

Esta función es parecida a la pasada, la diferencia de esta es que elimina al inicio de la posición de la lista, esta función se llama Eliminar Inicio, y como ya lo había comentado es parecida a la función pasada pero a diferencia de este cuando verifica que si esta llena la lista manda un aviso al usuario de que esta llena, y si no eliminara el elemento , per antes de hacer este último, verificara si la lista es igual a 1, si es así este llamara la función de Eliminar Final, Y si no ya podrá eliminar el elemento de la primera posición.

```
TipoDato EliminarInicio(Nodo* lista){
    int aux=1, cont=0;
    TipoDato rpd;
    if(lista->numelm==cont){
        printf("Tu lista esta vacia\n");
    }
    else if(lista->numelm==aux){
        rpd=EliminarFinal(lista);
        return rpd;
    }
    rpd=lista->dato[0];
    do{
        lista->dato[cont]=lista->dato[aux];
        cont++;
        aux++;
    }while(aux!=lista->numelm);
    lista->numelm--;
    lista->dato[lista->numelm]=COMPV;

    return rpd;
}
```


Esta Función se llama Insertar CPI (se refiere Con la Posición Indicada), este lo que hace es ir a la posición que quiere ir al usuario e inserta un nuevo elemento, pero antes de hacer eso, primero verifica si la lista esta vacía, si es así, manda un mensaje al usuario de que la lista esta vacía, si no esté hará otra verificación sin la posición es menor que uno, si este cumple con la condición le saldrá un mensaje de que la posición que solicito no es válida al insertar al inicio de la lista, y si no cumple este hará otra verificación de que si la posición es mayor que la lista, si es así este le saldrá un mensaje que la posición que la posición que solicito no es válida al insertar al final de la lista, si no es así este ya podrá ingresar a la posición que pidió el usuario.

```
void InsertarCPI(Nodo *lista, TipoData elemento, int posicion){
    int aux, apy;
    if(lista->numelm==0){
        printf("Tu lista esta vacia\n");
        return;
    }
    else if(posicion<1){
        printf("La posicion que solicito no es valida pruebe con la opcion de insertar al inicio de la lista\n");
        return;
    }
    else if(posicion>=lista->numelm){
        printf("La posicion que solicito no es valida pruebe con la opcion de insertar al final de la lista\n");
        return;
    }
    else
        lista->numelm++;
    aux=lista->dato[posicion];
    lista->dato[posicion]=elemento;
    posicion++;
    do{
        apy=lista->dato[posicion];
        lista->dato[posicion]=aux;
        aux=apy;
        posicion++;
    }while(posicion!=lista->numelm);
    return;
}
```

Esta función es igual al anterior solo que en ves de insertar un elemento lo va a eliminar, esta función se llama Eliminar CPI

```
TipoData EliminarCPI(Nodo *lista, int posicion){
    int aux, cont;
    TipoData rpd;
    if(lista->numelm==0){
        printf("Tu lista esta vacia\n");
    }
    else if(posicion<1){
        printf("La posicion que solicito no es valida pruebe con la opcion de eliminar al inicio de la lista\n");
    }
    else if(posicion>=lista->numelm){
        printf("La posicion que solicito no es valida pruebe con la opcion de eliminar al final de la lista\n");
    }
    else
        cont=posicion;
    posicion--;
    rpd=lista->dato[posicion];
    do{
        aux=lista->dato[cont];
        lista->dato[posicion]=aux;
        cont++;
        posicion++;
    }while(cont!=lista->numelm);
    lista->numelm--;
    return rpd;
}
```

Y al momento de correr este programa saldrá:

```
Bienvenido al programa de dos listas
Insertar un elemento al inicio de la lista.....[1]
Insertar un elemento al final de la lista.....[2]
Insertar un elemento en un punto intermedio dentro del dominio de la lista...[3]
Eliminar un elemento al inicio de la lista.....[4]
Eliminar un elemento al final de la lista.....[5]
Eliminar un elemento en un punto intermedio dentro del dominio de la lista...[6]
Vaciar lista.....[7]
Salir del programa.....[8]

Primer lista:
Tu lista esta vacia

El valor eliminado de la primera lista fue: 4194432

Por favor escoja una opcion del menu:
```

Programa 15

“Escribir un programa para hacer una implementación del TAD lista lineal vinculada.”

1) Declaración de ficheros para utilizar las distintas herramientas dentro del programa, se puede ver la declaración del nodo que se va a recorrer, también son visibles los prototipos de función.

```
15.c
1 //Programa para implementar un TAD de Lista lineal vinculada
2 #include<stdio.h>
3 #include<conio.h>
4 #include<malloc.h>
5 #include<stdlib.h>
6
7 struct nodo{
8     int dato;
9     struct nodo *sig;
10 };
11
12 //-----PROTOTIPOS DE FUNCION-----
13 struct nodo *inicio = NULL;
14 struct nodo *crear_list(struct nodo *);
15 struct nodo *mostrar(struct nodo *);
16 struct nodo *insertar_inicio(struct nodo *);
17 struct nodo *insertar_final(struct nodo *);
18 struct nodo *insertar_antes(struct nodo *);
19 struct nodo *insertar_desp(struct nodo *);
20 struct nodo *eliminar_inicio(struct nodo *);
21 struct nodo *eliminar_final(struct nodo *);
22 struct nodo *eliminar_node(struct nodo *);
23 struct nodo *eliminar_despues(struct nodo *);
24 struct nodo *eliminar_lista(struct nodo *);
```

2) Se imprime la serie de opciones así como da preferencia de ingresar que operación queremos realizar

```

26 int main(void){
27     int op;
28     do{
29         printf("\n*****OPERACIONES DE LA LISTA ENLAZADA*****");
30         printf("\n1. CREAR LISTA");
31         printf("\n2.MOSTRAR LISTA");
32         printf("\n3.AGREGAR NODO AL INICIO");
33         printf("\n4.AGREGAR NODO AL FINAL");
34         printf("\n5.AGREGAR UN NODO ANTES DE UN NODO DADO");
35         printf("\n6.AGREGAR UN NODO DESPUES DE UN NODO DADO");
36         printf("\n7.ELIMINAR UN NODO DEL INICIO");
37         printf("\n8.ELIMINAR UN NODO DEL FINAL");
38         printf("\n9.ELIMINAR UN NODO DADO");
39         printf("\n10.ELIMINAR UN NODO DESPUES DE UN NODO DADO");
40         printf("\n11.ELIMINAR LA LISTA COMPLETA");
41         printf("\n12.SALIR");
42         printf("\n\nIngresa la operacion que deseas elegir: ");
43         scanf("%d",&op);

```

3) Casos posibles para la opción que se ingresó

```

44     switch(op)
45     {
46
47         case 1:
48             inicio = crear_list(inicio);
49             printf("\nLISTA ENLAZADA CREADA");
50             break;
51
52         case 2:
53             inicio = mostrar(inicio);
54             break;
55
56         case 3:
57             inicio = insertar_inicio(inicio);
58             break;
59
60         case 4:
61             inicio = insertar_final(inicio);
62             break;
63
64         case 5:
65             inicio = insertar_antes(inicio);
66             break;
67
68         case 6:
69             inicio = insertar_desp(inicio);
70             break;
71
72         case 7:
73             inicio = eliminar_inicio(inicio);
74             break;
75
76         case 8:
77             inicio = eliminar_final(inicio);
78             break;
79
80         case 9:
81             inicio = eliminar_node(inicio);
82             break;
83
84         case 10:
85             inicio = eliminar_despues(inicio);
86             break;
87
88         case 11:
89             inicio = eliminar_lista(inicio);
90             printf("\nLISTA ENLAZADA ELIMINADA");
91             break;
92     }
93
94     while(op != 12);
95     getch();
96     return 0;
97 }

```

4) Función para crear la lista con memoria dinámica con ayuda de una estructura apuntado a la posición inicial revisando si la lista está vacía.

```
100 //Funcion para crear la lista
101 struct nodo *crear_list(struct nodo *inicio){
102     struct nodo *nuevo_nodo, *ptr;
103     int num;
104     printf("\nIngrese -1 para finalizar");
105     printf("\nIngrese su dato: ");
106     scanf("%d",&num);
107     while(num != -1){
108         nuevo_nodo = (struct nodo*)malloc(sizeof(struct nodo));
109         nuevo_nodo->dato = num;
110         if(inicio == NULL){
111             nuevo_nodo->sig = NULL;
112             inicio = nuevo_nodo;
113         }
114         else{
115             ptr=inicio;
116             while(ptr->sig!=NULL){
117                 ptr=ptr->sig;
118                 ptr->sig=nuevo_nodo;
119                 nuevo_nodo->sig=NULL;
120             }
121         }
122         printf("\nIngrese su dato: ");
123         scanf("%d",&num);
124     }
125     return inicio;
126 }
```

5) Función para mostrar los datos de la lista revisando si se encuentra apuntando a NULL para hacer la búsqueda e impresión

```
128 //Funcion para mostrar los datos de la lista
129 struct nodo *mostrar(struct nodo *inicio){
130     struct nodo *ptr;
131     ptr=inicio;
132     while(ptr != NULL){
133         printf("\t %d",ptr->dato);
134         ptr=ptr->sig;
135     }
136     return inicio;
137 }
```

6) Función que permite insertar un nodo al inicio

```
139 //Funcion para insertar un nodo al inicio
140 struct nodo *insertar_inicio(struct nodo *inicio){
141     struct nodo *nuevo_nodo;
142     int num;
143     printf("\nIngrese su dato: ");
144     scanf("%d",&num);
145     nuevo_nodo=(struct nodo*)malloc(sizeof(struct nodo));
146     nuevo_nodo->dato=num;
147     nuevo_nodo->sig=inicio;
148     inicio=nuevo_nodo;
149     return inicio;
150 }
```

7) Función para insertar un nodo al final

```
153 //Funcion para insertar un nodo al final
154 struct nodo *insertar_final(struct nodo *inicio){
155     struct nodo *ptr, *nuevo_nodo;
156     int num;
157     printf("\nIngrese su dato: ");
158     scanf("%d",&num);
159     nuevo_nodo=(struct nodo*)malloc(sizeof(struct nodo));
160     nuevo_nodo->dato=num;
161     nuevo_nodo->sig=NULL;
162     ptr=inicio;
163     while(ptr->sig != NULL){
164         ptr = ptr->sig;
165         ptr->sig=nuevo_nodo;
166     }
167     return inicio;
168 }
```

8) Función para insertar un nodo antes de uno dado.

```
171 //Funcion para insertar un nodo antes de uno dado
172 struct nodo *insertar_antes(struct nodo *inicio){
173     struct nodo *nuevo_nodo, *ptr, *preptr;
174     int num, val;
175     printf("\nIngrese su dato: ");
176     scanf("%d",&num);
177     printf("\nIngrese el valor anterior a donde el dato no ha sido insertado: ");
178     scanf("%d",&val);
179     nuevo_nodo=(struct nodo *)malloc(sizeof(struct nodo));
180     nuevo_nodo->dato=num;
181     ptr=inicio;
182     while(ptr->dato!=val){
183         preptr =ptr;
184         ptr=ptr->sig;
185     }
186     preptr->sig=nuevo_nodo;
187     nuevo_nodo->sig=ptr;
188     return inicio;
189 }
```

9) Función para insertar un nodo después de uno dado

```
191 //Funcion para insertar un nodo despues de uno dado
192 struct nodo *insertar_desp(struct nodo *inicio){
193     struct nodo *nuevo_nodo, *ptr, *preptr;
194     int num, val;
195     printf("\nIngrese su dato: ");
196     scanf("%d",&num);
197     printf("\nIngrese el valor posterior a donde se insertara el dato: ");
198     scanf("%d",&val);
199     nuevo_nodo = (struct nodo *)malloc(sizeof(struct nodo));
200     nuevo_nodo->dato=num;
201     ptr=inicio;
202     preptr=ptr;
203     while(preptr->dato!=val){
204         preptr =ptr;
205         ptr=ptr->sig;
206     }
207     preptr->sig=nuevo_nodo;
208     nuevo_nodo->sig=ptr;
209     return inicio;
210 }
```

10) Función para eliminar un nodo del inicio

```
213 //Funcion para eliminar un nodo del inicio
214 struct nodo *eliminar_inicio(struct nodo *inicio){
215     struct nodo *ptr;
216     ptr = inicio;
217     inicio = inicio->sig;
218     free(ptr);
219     return inicio;
220 }
```

11) Función para eliminar un nodo al final

```
222 //Funcion para eliminar un nodo al final
223 struct nodo *eliminar_final(struct nodo *inicio){
224     struct nodo *ptr, *preptr;
225     ptr = inicio;
226     while(ptr->sig != NULL){
227         preptr = ptr;
228         ptr = ptr -> sig;
229     }
230     preptr -> sig = NULL;
231     free(ptr);
232     return inicio;
233 }
```

12) Función para eliminar el nodo que se ingrese.

```
236 //Funcion para eliminar nodo
237 struct nodo *eliminar_node(struct nodo *inicio){
238     struct nodo *ptr, *preptr;
239     int val;
240     printf("\nIngresa el valor del nodo que deseas eliminar: ");
241     scanf("%d",&val);
242     ptr = inicio;
243     if(ptr->dato == val){
244         inicio= eliminar_inicio(inicio);
245         return inicio;
246     }
247     else{
248         while(ptr->dato != val){
249             preptr =ptr;
250             ptr =ptr ->sig;
251         }
252         preptr->sig =ptr-> sig;
253         free(ptr);
254         return inicio;
255     }
256 }
```

13) Función para eliminar un nodo después de que el tanque se lleno

```
259 //Funcion para eliminar un nodo despues del que se haya ingresddo
260 struct nodo *eliminar_despues(struct nodo *inicio){
261     struct nodo *ptr, *preptr;
262     int val;
263     printf("\nIngresa el valor despues del nodo que se desea eliminar: ");
264     scanf("%d",&val);
265     ptr = inicio;
266     preptr = ptr;
267     while(preptr->dato != val){
268         preptr = ptr;
269         ptr = ptr -> sig;
270     }
271     preptr->sig=ptr->sig;
272     free(ptr);
273     return inicio;
274 }
```

14) Función para eliminar toda la lista

```
276 //Funcion para eliminar toda la lista
277 struct nodo *eliminar_lista(struct nodo *inicio){
278     struct nodo *ptr;
279     if(inicio != NULL){
280         ptr = inicio;
281         while(ptr != NULL){
282             printf("\n %d va a ser eliminado", ptr->dato);
283             inicio = eliminar_inicio(ptr);
284             ptr = inicio;
285         }
286     }
287     return inicio;
288 }
```


15) A continuación se muestra la consola con el ingreso de operaciones

```
*****OPERACIONES DE LA LISTA ENLAZADA*****
1. CREAM LISTA
2.MOSTRAR LISTA
3.AGREGAR NODO AL INICIO
4.AGREGAR NODO AL FINAL
5.AGREGAR UN NODO ANTES DE UN NODO DADO
6.AGREGAR UN NODO DESPUES DE UN NODO DADO
7.ELIMINAR UN NODO DEL INICIO
8.ELIMINAR UN NODO DEL FINAL
9.ELIMINAR UN NODO DADO
10.ELIMINAR UN NODO DESPUES DE UN NODO DADO
11.ELIMINAR LA LISTA COMPLETA
12.SALIR

Ingresa la operacion que deseas elegir: 1

Ingresa -1 para finalizar
Ingresa su dato: 4

Ingresa su dato: -1

LISTA ENLAZADA CREADA
*****OPERACIONES DE LA LISTA ENLAZADA*****
1. CREAM LISTA
2.MOSTRAR LISTA
3.AGREGAR NODO AL INICIO
4.AGREGAR NODO AL FINAL
5.AGREGAR UN NODO ANTES DE UN NODO DADO
6.AGREGAR UN NODO DESPUES DE UN NODO DADO
7.ELIMINAR UN NODO DEL INICIO
8.ELIMINAR UN NODO DEL FINAL
9.ELIMINAR UN NODO DADO
10.ELIMINAR UN NODO DESPUES DE UN NODO DADO
11.ELIMINAR LA LISTA COMPLETA
12.SALIR

Ingresa la operacion que deseas elegir: 3

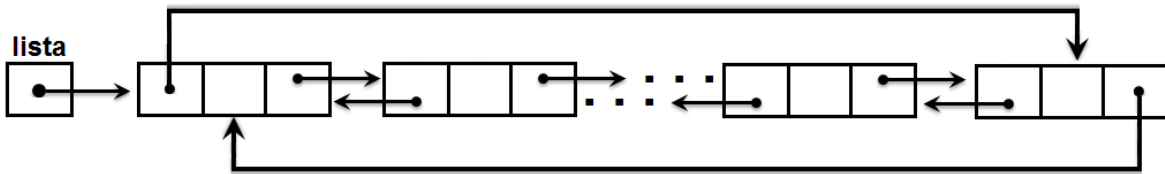
Ingresa su dato: 8
```

Programa 16

“Escribir un programa para hacer una implementación del TAD lista doblemente vinculada circular.”

Antes de comenzar a describir el funcionamiento del programa número 16 que lleva por nombre “Implementación del TAD lista doblemente vinculada circular” perteneciente a la unidad de aprendizaje titulada “Estructura de Datos” deberemos explicar las características principales de este tipo de TAD lista, con el fin de corroborar que el programa pueda ser considerado como tal.

El TAD de lista doblemente vinculada circular consiste en una lista lineal en la que cada elemento tiene dos enlaces, uno al nodo siguiente y otro al nodo anterior. Esto permite leer la lista en cualquier dirección. Asimismo su último elemento se enlaza con el primero, logrando acceder a cualquier elemento de la lista desde cualquier punto dado.



Son tres las operaciones básicas que podemos realizar con una lista doblemente enlazada circular:

- Insertar.- comprende varios casos: a) insertar un elemento al principio de la lista, b) insertar un elemento entre otros dos y c) insertar un elemento al final.
- Borrar.- comprende los siguientes casos: a) borrar el primer elemento de la lista, b) borrar elemento cualquiera entre otros dos y c) borrar el último elemento de la lista.
- Ver.- recorre e imprime los elementos de la lista.

Una vez dicho esto, comenzaremos a explicar mediante un ejemplo cada una de las opciones y propiedades que presenta nuestro programa.

En primera instancia lograremos observar los archivos de cabecera necesarios para obtener el correcto funcionamiento de las diversas herramientas utilizadas a lo largo del programa.

Posteriormente y en base a la recomendación del profesor haremos uso de typedef para el tipo de dato que guardaremos en cada nodo de la lista.

Esta acción tiene como propósito el poder realizar un cambio por parte del elemento solicitado al usuario de forma más sencilla y rápida. Con lo que respecta a la estructura del nodo y en base a una de las características principales del TAD lista doblemente vinculada circular, podemos notar que cuenta con dos apuntadores de tipo nodo, uno al nodo siguiente y otro al nodo anterior, además de contar con un indicador de posición para no perder la secuencia de los datos requerida por el usuario;

Por ultimo tenemos la estructura que nos ayudara a no perder nuestra lista pese a los cambios realizados en la misma, siendo el apuntador a nodo el punto de inicio, además que obtendremos un mayor control de la secuencia y cantidad de nodos mediante el uso de la variable nelementos (Número de elementos).

```
16.c
1  #include <string.h>
2  #include <stdio.h>
3  #include <conio.h>
4
5  typedef int TipoDato;
6
7  typedef struct DPST{
8      TipoDato dato;
9      int posicion;
10     struct DPST *next;
11     struct DPST *before;
12 }Nodo;
13
14 typedef struct INL{
15     int nelementos;
16     struct DPST *PN;
17 }Indicador;
18
```

```
19  //////////////////////////////////////
20
21  Indicador* CrearLista(Indicador *lista);
22
23  Nodo* CrearNodo(TipoDato elemento);
24
25  void InsertarInicio(Indicador *lista, TipoDato elemento);
26
27  void InsertarFinal(Indicador *lista, TipoDato elemento);
28
29  void InsertarCPI(Indicador *lista, TipoDato elemento, int PP);
30
31  TipoDato EliminarInicio(Indicador *lista);
32
33  TipoDato EliminarFinal(Indicador *lista);
34
35  TipoDato EliminarCPI(Indicador *lista, int PP);
36
37  void EliminarLista(Indicador *lista);
38
39  void ImprimirLista(Indicador *lista);
40
41  void ImprimirPosiciones(Indicador *lista);
42
43  void ImprimirNelementos(Indicador *lista);
44
45  //////////////////////////////////////
46
```

Posteriormente lograremos observar los encabezados de todas las funciones utilizadas para este programa, destacando aquellas funciones las cuales llevan por nombre las tres operaciones básicas que podemos realizar con una lista doblemente enlazada circular.

```

46
47 int main(void){
48     Indicador *Lista;
49     Lista=NULL;
50     Lista=CrearLista(Lista);
51
52     int elemento, posicion, opcion;
53     TipoDato eliminado;
54

```

Después encontraremos a la función main, donde la primera parte cuenta con las declaraciones de las variables a utilizar en las opciones que presenta el menú del programa.

```

47 int main(void){
48     Indicador *Lista;
49     Lista=NULL;
50     Lista=CrearLista(Lista);
51
52     int elemento, posicion, opcion;
53     TipoDato eliminado;
54
55
56     do{
57         system("cls");
58         printf("\nBienvenido al programa de lista\n");
59         printf("Insertar un elemento al inicio de la lista.....[1]\n");
60         printf("Insertar un elemento al final de la lista.....[2]\n");
61         printf("Insertar un elemento en un punto intermedio dentro del dominio de la lista...[3]\n");
62         printf("Eliminar un elemento al inicio de la lista.....[4]\n");
63         printf("Eliminar un elemento al final de la lista.....[5]\n");
64         printf("Eliminar un elemento en un punto intermedio dentro del dominio de la lista...[6]\n");
65         printf("Vaciar lista.....[7]\n");
66         printf("Salir del programa.....[8]\n");
67
68         printf("\nTu lista:\n");
69         ImprimirLista(Lista);
70         ImprimirNelementos(Lista);
71         printf("\nEl valor eliminado fue: %d\n", eliminado);
72
73         printf("\nPor favor escoja una opcion del menu:\n");
74         scanf("%d",&opcion);
75
76         if(opcion==1){
77             printf("Introduzca el dato de su preferencia\n");
78             scanf("%d",&elemento);
79             InsertarInicio(Lista,elemento);
80         }
81         else if(opcion==2){
82             printf("Introduzca el dato de su preferencia\n");
83             scanf("%d",&elemento);
84             InsertarFinal(Lista,elemento);
85         }
86         else if(opcion==3){
87             printf("Introduzca el dato de su preferencia\n");
88             scanf("%d",&elemento);
89             printf("Introduzca la posicion de su preferencia\n");
90             scanf("%d",&posicion);
91             InsertarCPI(Lista,elemento,posicion);
92         }
93     }

```

```

92     else if(opcion==4){
93         eliminado=EliminarInicio(Lista);
94     }
95     else if(opcion==5){
96         eliminado=EliminarFinal(Lista);
97     }
98     else if(opcion==6){
99         printf("Introduzca la posicion de su preferencia\n");
100        scanf("%d",&posicion);
101        eliminado=EliminarCPI(Lista,posicion);
102    }
103    else if(opcion==7){
104        EliminarLista(Lista);
105    }
106    else if(opcion==8){
107        printf("Nos vemos\n");
108        printf("\n");
109    }
110    else
111        printf("Opcion invalida\n");
112    printf("\n");
113    }while(opcion!=8);
114
115    return 0;
116
117
118 //////////////////////////////////////

```

A partir de este momento y con el fin de lograr una mejor comprensión del programa y demostrar su funcionalidad, haremos uso de ejemplos mediante el compilador para cada una de las funciones que presenta la función main.

```

system("cls");
printf("\nBienvenido al programa de lista\n");
printf("Insertar un elemento al inicio de la lista.....[1]\n");
printf("Insertar un elemento al final de la lista.....[2]\n");
printf("Insertar un elemento en un punto intermedio dentro del dominio de la lista...[3]\n");
printf("Eliminar un elemento al inicio de la lista.....[4]\n");
printf("Eliminar un elemento al final de la lista.....[5]\n");
printf("Eliminar un elemento en un punto intermedio dentro del dominio de la lista...[6]\n");
printf("Vaciar lista.....[7]\n");
printf("Salir del programa.....[8]\n");

printf("\nTu lista:\n");
ImprimirLista(Lista);
ImprimirNelementos(Lista);
printf("\nEl valor eliminado fue: %d\n", eliminado);

printf("\nPor favor escoja una opcion del menu:\n");
scanf("%d",&opcion);

```

Al no tener datos en la lista, las impresiones por parte del menú serán las siguientes:

```

C:\Users\alber_06a8k\Desktop\16.exe
Bienvenido al programa de lista
Insertar un elemento al inicio de la lista.....[1]
Insertar un elemento al final de la lista.....[2]
Insertar un elemento en un punto intermedio dentro del dominio de la lista...[3]
Eliminar un elemento al inicio de la lista.....[4]
Eliminar un elemento al final de la lista.....[5]
Eliminar un elemento en un punto intermedio dentro del dominio de la lista...[6]
Vaciar lista.....[7]
Salir del programa.....[8]

Tu lista:
Tu lista se encuentra vacia

Numero de elementos en tu lista: 0

El valor eliminado fue: 0

Por favor escoja una opcion del menu:

```

En primera instancia destacaremos la funcion “ImprimirLista” la cual cumple el papel de la operación basica del TAD lista doblemente vinculada circular “Ver”.

```

136 void ImprimirLista(Indicador *lista){
137     Nodo *imp;
138     if(lista->nelementos==0){
139         printf("Tu lista se encuentra vacia\n");
140         return;
141     }
142     else
143     imp=lista->PN;
144     do{
145         printf("%d ",imp->dato);
146         imp=imp->next;
147     }while(imp!=lista->PN);
148     printf("\n");
149     return;
150 }

```

Dicha función en términos generales recibe la lista mediante el apuntador de estructura indicador, asimismo crearemos un apuntador a nodo con el fin de no perder el punto de inicio al momento de hacer la impresión del elemento que contiene cada nodo. Por parte del elemento eliminado, al no tener un valor definido nos arroja por defecto un 0.

Al seleccionar la primera opción haremos uso de la primera función basada en uno de los casos que comprende la operación básica de insertar.

```

C:\Users\valber_06a8k\Desktop\16.exe
Bienvenido al programa de lista
Insertar un elemento al inicio de la lista.....[1]
Insertar un elemento al final de la lista.....[2]
Insertar un elemento en un punto intermedio dentro del dominio de la lista...[3]
Eliminar un elemento al inicio de la lista.....[4]
Eliminar un elemento al final de la lista.....[5]
Eliminar un elemento en un punto intermedio dentro del dominio de la lista...[6]
Vaciar lista.....[7]
Salir del programa.....[8]

Tu lista:
Tu lista se encuentra vacia

Numero de elementos en tu lista: 0

El valor eliminado fue: 0

Por favor escoja una opcion del menu:
1
Introduzca el dato de su preferencia

```

```

if(opcion==1){
    printf("Introduzca el dato de su preferencia\n");
    scanf("%d",&elemento);
    InsertarInicio(Lista,elemento);
}

```

En primera instancia además de lograr observar los datos que solicita la función para su correcto funcionamiento.

```

188 void InsertarInicio(Indicador *lista, TipoDato elemento){
189     Nodo *New;
190     if(lista->nelementos==0){
191         lista->PN=CrearNodo(elemento);
192         lista->nelementos++;
193         (lista->PN)->next=lista->PN;
194         (lista->PN)->before=lista->PN;
195         (lista->PN)->posicion=lista->nelementos;
196         return;
197     }
198     else
199     {
200         New=CrearNodo(elemento);
201         New->next=lista->PN;
202         New->before=(lista->PN)->before;
203         (New->before)->next=New;
204         (lista->PN)->before=New;
205         New->posicion=1;
206         do{
207             (lista->PN)->posicion++;
208             lista->PN=(lista->PN)->next;
209         }while(lista->PN!=New);
210         lista->nelementos++;
211         return;
212     }
}

```

Después si la lista se encuentra vacía apuntaremos al nodo mediante el punto de inicio. En cambio si ya se tiene como mínimo un elemento en la lista, utilizaremos los apuntadores del primer nodo como referencia para hacer el cambio de posición pero de igual forma el punto de inicio terminara apuntando al nuevo nodo.

```

Tu lista:
7

Numero de elementos en tu lista: 1

El valor eliminado fue: 0

Por favor escoja una opcion del menu:

```

Asimismo deberemos incrementar la posición de los nodos posteriores para no perder la secuencia y mantener un control en los datos.

Una vez introducido el dato de nuestra preferencia se presentara un cambio en el menú. Como logramos observar al imprimir de nueva cuenta la lista, ahora aparece el valor introducido hace unos momentos, además que el número de elementos se incrementa.

Ahora volveremos a seleccionar la primera opción para corroborar la inserción al principio de la lista.

```
C:\Users\alber_06a8k\Desktop\16.exe

Bienvenido al programa de lista
Insertar un elemento al inicio de la lista.....[1]
Insertar un elemento al final de la lista.....[2]
Insertar un elemento en un punto intermedio dentro del dominio de la lista...[3]
Eliminar un elemento al inicio de la lista.....[4]
Eliminar un elemento al final de la lista.....[5]
Eliminar un elemento en un punto intermedio dentro del dominio de la lista...[6]
Vaciar lista.....[7]
Salir del programa.....[8]

Tu lista:
7

Numero de elementos en tu lista: 1

El valor eliminado fue: 0

Por favor escoja una opcion del menu:
1
Introduzca el dato de su preferencia
6
```

```
C:\Users\alber_06a8k\Desktop\16.exe

Bienvenido al programa de lista
Insertar un elemento al inicio de la lista.....[1]
Insertar un elemento al final de la lista.....[2]
Insertar un elemento en un punto intermedio dentro del dominio de la lista...[3]
Eliminar un elemento al inicio de la lista.....[4]
Eliminar un elemento al final de la lista.....[5]
Eliminar un elemento en un punto intermedio dentro del dominio de la lista...[6]
Vaciar lista.....[7]
Salir del programa.....[8]

Tu lista:
6 7

Numero de elementos en tu lista: 2

El valor eliminado fue: 0

Por favor escoja una opcion del menu:
```

Ahora somos capaces de observar cómo se cumplió la operación de insertar al inicio de la lista, ya que no se encuentra vacía. Ahora comprobaremos la segunda opción.

```
C:\Users\alber_06a8k\Desktop\16.exe

Bienvenido al programa de lista
Insertar un elemento al inicio de la lista.....[1]
Insertar un elemento al final de la lista.....[2]
Insertar un elemento en un punto intermedio dentro del dominio de la lista...[3]
Eliminar un elemento al inicio de la lista.....[4]
Eliminar un elemento al final de la lista.....[5]
Eliminar un elemento en un punto intermedio dentro del dominio de la lista...[6]
Vaciar lista.....[7]
Salir del programa.....[8]

Tu lista:
6 7

Numero de elementos en tu lista: 2

El valor eliminado fue: 0

Por favor escoja una opcion del menu:
2
Introduzca el dato de su preferencia
8
```



```

213 void InsertarFinal(Indicador *lista, TipoDato elemento){
214     Nodo *New;
215     if(lista->nelementos==0){
216         lista->PN=CrearNodo(elemento);
217         lista->nelementos++;
218         (lista->PN)->next=lista->PN;
219         (lista->PN)->before=lista->PN;
220         (lista->PN)->posicion=lista->nelementos;
221         return;
222     }
223     else
224     {
225         New=CrearNodo(elemento);
226         New->before=(lista->PN)->before;
227         New->next=lista->PN;
228         (New->before)->next=New;
229         (lista->PN)->before=New;
230         lista->nelementos++;
231         New->posicion=lista->nelementos;
232         return;
233     }
234 }

```

Como podemos observar, esta función comparte similitudes con la función de InsertarInicio ya que si se presenta la lista vacía, el punto de inicio apuntara al nodo creado, pero en el caso de que la lista posea como mínimo otro dato, se hará uso de los apuntes del nodo ubicado en el punto de inicio ya que este tipo de TAD lista nos ofrece la opción de evitar usar un punto final debido a su característica circular. El cambio en el compilador nos confirmara el cumplimiento de la operación de insertar al final de la lista, gracias al uso de esta función.

```

C:\Users\valber_06a8k\Desktop\16.exe
Bienvenido al programa de lista
Insertar un elemento al inicio de la lista.....[1]
Insertar un elemento al final de la lista.....[2]
Insertar un elemento en un punto intermedio dentro del dominio de la lista...[3]
Eliminar un elemento al inicio de la lista.....[4]
Eliminar un elemento al final de la lista.....[5]
Eliminar un elemento en un punto intermedio dentro del dominio de la lista...[6]
Vaciar lista.....[7]
Salir del programa.....[8]

Tu lista:
6 7 8

Numero de elementos en tu lista: 3

El valor eliminado fue: 0

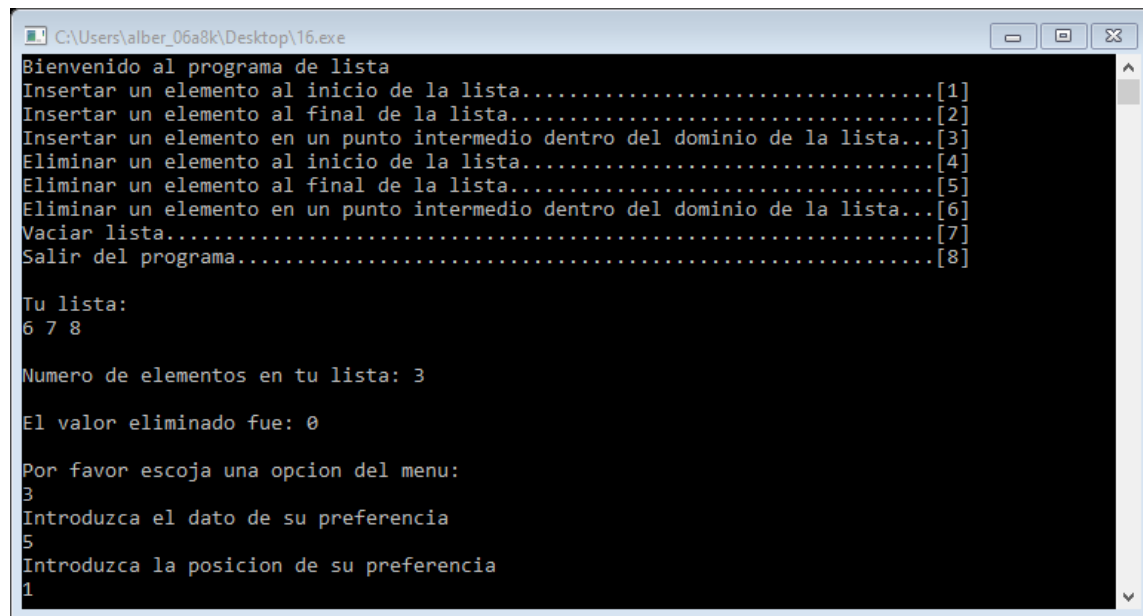
Por favor escoja una opcion del menu:

```

Ahora, corresponde la revisión de la opción de InsertarCPI (Insertar en Cualquier Parte Intermedia), en base a las características de la operación insertar un elemento entre otros dos, por parte de lista doblemente enlazada circular. Para esta función en particular se debe hacer una aclaración con respecto a la inserción del nuevo dato. Dependiendo de la ubicación donde queremos colocar nuestro elemento, deberemos escribir en el programa un número anterior. Por ejemplo, en este caso quiero

introducir el número 5 en la posición 2 de mi arreglo. Deberé introducir el número 1 al momento que me pregunte por la posición de mi preferencia.

```
else if(opcion==3){
    printf("Introduzca el dato de su preferencia\n");
    scanf("%d",&elemento);
    printf("Introduzca la posicion de su preferencia\n");
    scanf("%d",&posicion);
    InsertarCPI(Lista,elemento,posicion);
}
```



```
C:\Users\alber_06a8k\Desktop\16.exe
Bienvenido al programa de lista
Insertar un elemento al inicio de la lista.....[1]
Insertar un elemento al final de la lista.....[2]
Insertar un elemento en un punto intermedio dentro del dominio de la lista...[3]
Eliminar un elemento al inicio de la lista.....[4]
Eliminar un elemento al final de la lista.....[5]
Eliminar un elemento en un punto intermedio dentro del dominio de la lista...[6]
Vaciar lista.....[7]
Salir del programa.....[8]

Tu lista:
6 7 8

Numero de elementos en tu lista: 3

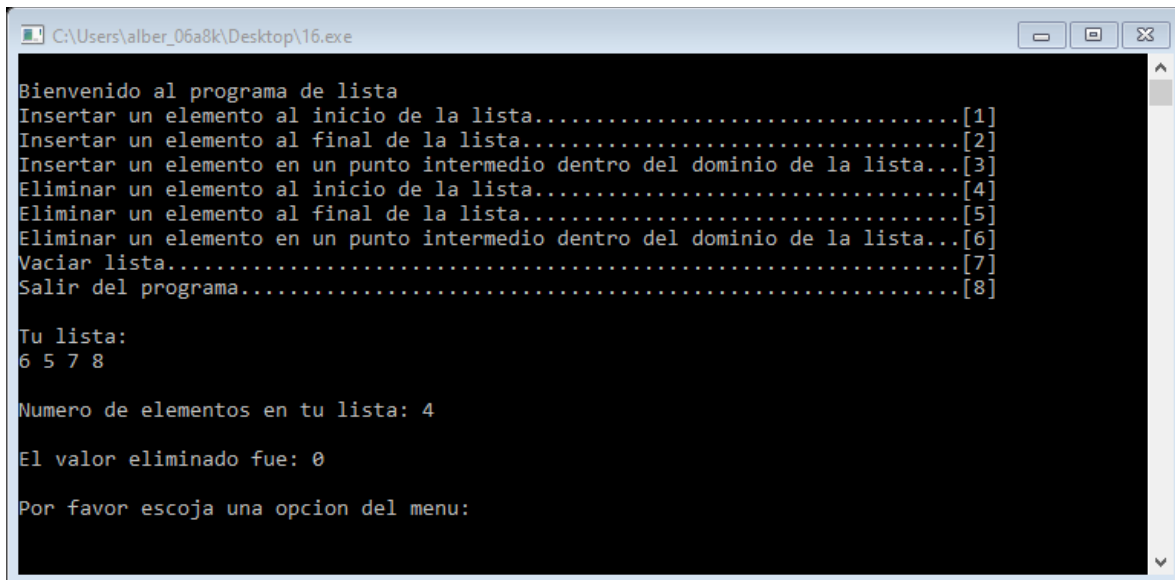
El valor eliminado fue: 0

Por favor escoja una opcion del menu:
3
Introduzca el dato de su preferencia
5
Introduzca la posicion de su preferencia
1
```

```
289 void InsertarCPI(Indicador *lista, TipoDato elemento, int PP){
290     Nodo *New, *aux;
291     if(lista->nelementos==0){
292         printf("Tu lista se encuentra vacia\n");
293         return;
294     }
295     else if(PP>=lista->nelementos){
296         printf("La posicion que solicito no es valida pruebe con la opcion de insertar al final de la lista\n");
297         return;
298     }
299     else if(PP<1){
300         printf("La posicion que solicito no es valida pruebe con la opcion de insertar al inicio de la lista\n");
301         return;
302     }
303     else
304     New=CrearNodo(elemento);
305     aux=lista->PN;
306     do{
307         aux=aux->next;
308     }while(aux->posicion!=PP);
309     New->posicion=PP;
310     New->before=aux;
311     New->next=aux->next;
312     (aux->next)->before=New;
313     aux->next=New;
314     do{
315         New->posicion++;
316         New=New->next;
317     }while(New!=lista->PN);
318     lista->nelementos++;
319     return;
```

Al introducir el nuevo valor, primero se verificara que como mínimo la lista posea dos elementos, ya que se busca cumplir la condición "entre otros nodos", posteriormente mediante la ayuda de otro apuntador a nodo,

este buscara la posición seleccionada por el usuario, introduciendo el nuevo elemento en una posición posterior a la marcada con el usuario.



```
C:\Users\alber_06a8k\Desktop\16.exe

Bienvenido al programa de lista
Insertar un elemento al inicio de la lista.....[1]
Insertar un elemento al final de la lista.....[2]
Insertar un elemento en un punto intermedio dentro del dominio de la lista...[3]
Eliminar un elemento al inicio de la lista.....[4]
Eliminar un elemento al final de la lista.....[5]
Eliminar un elemento en un punto intermedio dentro del dominio de la lista...[6]
Vaciar lista.....[7]
Salir del programa.....[8]

Tu lista:
6 5 7 8

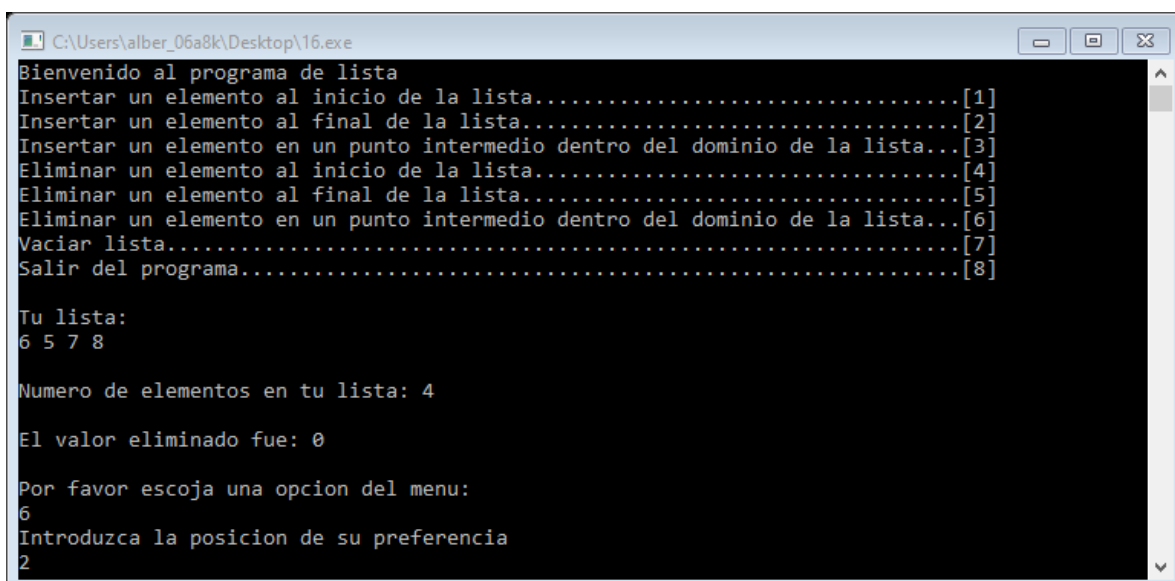
Numero de elementos en tu lista: 4

El valor eliminado fue: 0

Por favor escoja una opcion del menu:
```

Posteriormente y aprovechando nuestro ejemplo procederemos a demostrar la función de la opción EliminarCPI (Eliminar Cualquier Posición Intermedia). Para esto a diferencia de InsertarCPI, la posición seleccionada por el usuario mientras cumpla la condición "entre otros nodos" será la del elemento eliminado.

```
else if(opcion==6){
    printf("Introduzca la posicion de su preferencia\n");
    scanf("%d",&posicion);
    eliminado=EliminarCPI(Lista,posicion);
}
```



```
C:\Users\alber_06a8k\Desktop\16.exe

Bienvenido al programa de lista
Insertar un elemento al inicio de la lista.....[1]
Insertar un elemento al final de la lista.....[2]
Insertar un elemento en un punto intermedio dentro del dominio de la lista...[3]
Eliminar un elemento al inicio de la lista.....[4]
Eliminar un elemento al final de la lista.....[5]
Eliminar un elemento en un punto intermedio dentro del dominio de la lista...[6]
Vaciar lista.....[7]
Salir del programa.....[8]

Tu lista:
6 5 7 8

Numero de elementos en tu lista: 4

El valor eliminado fue: 0

Por favor escoja una opcion del menu:
6
Introduzca la posicion de su preferencia
2
```

Algo que deberemos destacar de las funciones “Eliminar” es el hecho de que cada una nos regresa el valor eliminado el cual se mostrara debajo del número de elementos en la lista.

```
322 TipoDato EliminarCPI(Indicador *lista, int PP){
323     Nodo *Nlib, *aux;
324     TipoDato rpd;
325     if(lista->nelementos==0){
326         printf("Tu lista se encuentra vacia\n");
327         return;
328     }
329     else if(PP>=lista->nelementos){
330         printf("La posicion que solicito no es valida pruebe con la opcion de eliminar el final de la lista\n");
331         return;
332     }
333     else if(PP<=1){
334         printf("La posicion que solicito no es valida pruebe con la opcion de eliminar el inicio de la lista\n");
335         return;
336     }
337     else
338         Nlib=lista->PN;
339     do{
340         Nlib=Nlib->next;
341     }while(Nlib->posicion!=PP);
342     rpd=Nlib->dato;
343     (Nlib->before)->next=Nlib->next;
344     (Nlib->next)->before=Nlib->before;
345     aux=Nlib->next;
346     do{
347         aux->posicion--;
348         aux=aux->next;
349     }while(aux!=lista->PN);
350     lista->nelementos--;
351     free(Nlib);
352     return rpd;
```

Para lograr esto, podemos observar que la función presenta un dato de tipo entero en su regreso, asimismo de forma similar a la función InsertarCPI, se encarga de verificar que la lista cuente con un mínimo de tres elementos. Posteriormente mediante la ayuda de un apuntador a nodo, este último se encargara de apuntar al nodo que se desea liberar por parte del usuario, asimismo otro apuntador a nodo llamado “aux” se encargara de apuntar al nodo que se encuentra en la posición posterior del elemento a liberar, con el fin de hacer los cambios pertinentes en sus apuntadores internos y de esta forma liberar la memoria sin provocar alguna perdida en la lista, además de corregir el valor de posición en los demás nodos.

```
C:\Users\alber_06a8k\Desktop\16.exe

Bienvenido al programa de lista
Insertar un elemento al inicio de la lista.....[1]
Insertar un elemento al final de la lista.....[2]
Insertar un elemento en un punto intermedio dentro del dominio de la lista...[3]
Eliminar un elemento al inicio de la lista.....[4]
Eliminar un elemento al final de la lista.....[5]
Eliminar un elemento en un punto intermedio dentro del dominio de la lista...[6]
Vaciar lista.....[7]
Salir del programa.....[8]

Tu lista:
6 7 8

Numero de elementos en tu lista: 3

El valor eliminado fue: 5

Por favor escoja una opcion del menu:
```

Clara prueba de esta accion sera la presentacion del valor eliminado en la posicion solicitada por el usuario, en el menu.

Después comprobaremos la función de EliminarInicio, al igual que las otras funciones de eliminar, esta regresara el elemento eliminado en el apartado del menú correspondiente para verificar que se realizo dicha accion.

```
C:\Users\alber_06a8k\Desktop\16.exe

Bienvenido al programa de lista
Insertar un elemento al inicio de la lista.....[1]
Insertar un elemento al final de la lista.....[2]
Insertar un elemento en un punto intermedio dentro del dominio de la lista...[3]
Eliminar un elemento al inicio de la lista.....[4]
Eliminar un elemento al final de la lista.....[5]
Eliminar un elemento en un punto intermedio dentro del dominio de la lista...[6]
Vaciar lista.....[7]
Salir del programa.....[8]

Tu lista:
6 7 8

Numero de elementos en tu lista: 3

El valor eliminado fue: 5

Por favor escoja una opcion del menu:
4
```

```

234 TipoDato EliminarInicio(Indicador *lista){
235     Nodo *Nlib;
236     TipoDato rpd;
237     if(lista->nelementos==0){
238         printf("Tu lista esta vacia\n");
239         return;
240     }
241     else if(lista->nelementos==1){
242         rpd=(lista->PN)->dato;
243         free(lista->PN);
244         lista->PN=NULL;
245         lista->nelementos--;
246         printf("Eliminaste el ultimo elemento de tu lista\n");
247         return rpd;
248     }
249     else
250     rpd=(lista->PN)->dato;
251     Nlib=lista->PN;
252     lista->PN=(lista->PN)->next;
253     (Nlib->before)->next=lista->PN;
254     (lista->PN)->before=Nlib->before;
255     lista->nelementos--;
256     do{
257         (lista->PN)->posicion--;
258         lista->PN=(lista->PN)->next;
259     }while((lista->PN)->posicion!=1);
260     free(Nlib);
261     return rpd;
262 }

```

Para esta funcion se consideran tres casos, el primero es si la lista se encuentra vacia, de tal forma que si llegara a cumplirse este no retornaria nada mas que la frase de advertencia. Para el segundo caso, se considera el eliminar el ultimo elemento de la lista, provocando que esta se encuentre vacia, soltando la señal de advertencia. Por ultimo al no cumplirse los dos anteriores casos, mediante la ayuda de un apuntador a nodo, se realiza el cambio de posiciones y la disminucion de las posiciones de cada nodo restante en la lista.

```

C:\Users\alber_06a8k\Desktop\16.exe
Bienvenido al programa de lista
Insertar un elemento al inicio de la lista.....[1]
Insertar un elemento al final de la lista.....[2]
Insertar un elemento en un punto intermedio dentro del dominio de la lista...[3]
Eliminar un elemento al inicio de la lista.....[4]
Eliminar un elemento al final de la lista.....[5]
Eliminar un elemento en un punto intermedio dentro del dominio de la lista...[6]
Vaciar lista.....[7]
Salir del programa.....[8]

Tu lista:
7 8

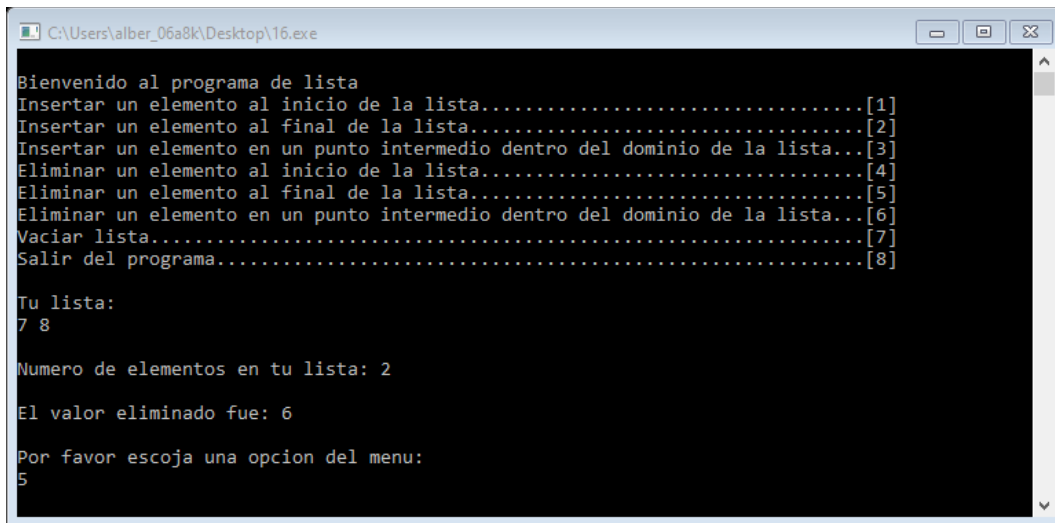
Numero de elementos en tu lista: 2

El valor eliminado fue: 6

Por favor escoja una opcion del menu:

```

Por ultimo tendremos la opción de “EliminarFinal”, la cual comparte muchas similitudes con la opción “EliminarInicio”.



```
C:\Users\alber_06a8k\Desktop\16.exe

Bienvenido al programa de lista
Insertar un elemento al inicio de la lista.....[1]
Insertar un elemento al final de la lista.....[2]
Insertar un elemento en un punto intermedio dentro del dominio de la lista...[3]
Eliminar un elemento al inicio de la lista.....[4]
Eliminar un elemento al final de la lista.....[5]
Eliminar un elemento en un punto intermedio dentro del dominio de la lista...[6]
Vaciar lista.....[7]
Salir del programa.....[8]

Tu lista:
7 8

Numero de elementos en tu lista: 2

El valor eliminado fue: 6

Por favor escoja una opcion del menu:
5
```

```
264 TipoDato EliminarFinal(Indicador *lista){
265     Nodo *Nlib;
266     TipoDato rpd;
267     if(lista->nelementos==0){
268         printf("Tu lista esta vacia\n");
269         return;
270     }
271     else if(lista->nelementos==1){
272         rpd=(lista->PN)->dato;
273         free(lista->PN);
274         lista->PN=NULL;
275         lista->nelementos--;
276         printf("Eliminaste el ultimo elemento de tu lista\n");
277         return rpd;
278     }
279     else
280     {
281         Nlib=(lista->PN)->before;
282         rpd=Nlib->dato;
283         (lista->PN)->before=Nlib->before;
284         (Nlib->before)->next=lista->PN;
285         lista->nelementos--;
286         free(Nlib);
287         return rpd;
288     }
289 }
```

Se consideran de nueva cuenta tres casos, de forma similar a la opción de EliminarInicio, por ende, el tercer caso resulta más de interés, ya que mediante el uso del apuntador de la posición anterior por parte del punto principal lograremos colocar el apuntador a nodo “Nlib” de forma más sencilla al dato que por nombre de la función se desea eliminar. Realizando los respectivos cambios con sus apuntadores y liberando el espacio de memoria. Observando dicho cambio mediante la devolución del valor entero eliminado, en la parte del menú correspondiente.

```
C:\Users\alber_06a8k\Desktop\16.exe

Bienvenido al programa de lista
Insertar un elemento al inicio de la lista.....[1]
Insertar un elemento al final de la lista.....[2]
Insertar un elemento en un punto intermedio dentro del dominio de la lista...[3]
Eliminar un elemento al inicio de la lista.....[4]
Eliminar un elemento al final de la lista.....[5]
Eliminar un elemento en un punto intermedio dentro del dominio de la lista...[6]
Vaciar lista.....[7]
Salir del programa.....[8]

Tu lista:
7

Numero de elementos en tu lista: 1

El valor eliminado fue: 8

Por favor escoja una opcion del menu:
```


Programa 17

“Hacer un programa que permita realizar operaciones entre dos listas que están implementadas de manera estática.”

Debido a las diversas similitudes con el programa 14, la descripción del programa 17 titulado “Operaciones entre dos listas que están implementadas de manera estática” perteneciente a la unidad de aprendizaje que lleva por nombre “Estructura de Datos” se basará únicamente en el apartado de las operaciones entre las dos listas, así como cambios de relevancia en el menú de opciones.

Primero podremos observar los archivos de cabecera necesarios para obtener el correcto funcionamiento de las diversas herramientas utilizadas a lo largo del programa. Asimismo, debido a que utilizaremos listas estáticas debemos definir un número máximo de elementos para la cadena de datos en cada nodo.

```
1  #include <string.h>
2  #include <stdio.h>
3  #include <conio.h>
4  #define MAX 20
5  #define COMPV 1
6
7  typedef int TipoDato;
8
9  typedef struct alm{
10     int numelm;
11     TipoDato dato[MAX];
12 }Nodo;
```

```
14 void CrearOVaciarLista(Nodo* Lista);
15
16 void ImpresionLista(Nodo* Lista);
17
18 void ImprimirNelementos(Nodo *lista);
19
20 void InsertarFinal(Nodo* Lista, TipoDato elemento);
21
22 void InsertarInicio(Nodo* lista, TipoDato elemento);
23
24 TipoDato EliminarFinal(Nodo* lista);
25
26 TipoDato EliminarInicio(Nodo* lista);
27
28 void InsertarCPI(Nodo *lista, TipoDato elemento, int posicion);
29
30 TipoDato EliminarCPI(Nodo *lista, int posicion);
31
32 void ConcatenarListas(Nodo *plista, Nodo *slista);
33
34 void BuscarComunes(Nodo *plista, Nodo *slista);
35
36 void CambiarDato(Nodo *plista, Nodo *slista);
```

Posteriormente lograremos observar los encabezados de todas las funciones utilizadas para este programa, destacando las últimas tres, ya que solicitan los apuntadores de ambas listas estáticas.

```
Nodo LU, LD, *Lista, *SL;
Lista=&LU;
SL=&LD;
CrearOVaciarLista(Lista);
CrearOVaciarLista(SL);
int posicion, opcion, nlista, nelista, ubicacion;
TipoDato elemento, eliminado;
```

Después tendremos la declaración de las variables a utilizar durante la mayor parte del programa, más en específico el main.

```
49 do{
50     system("cls");
51     printf("\nBienvenido al programa de dos listas\n");
52     printf("Insertar un elemento al inicio de la lista.....[1]\n");
53     printf("Insertar un elemento al final de la lista.....[2]\n");
54     printf("Insertar un elemento en un punto intermedio dentro del dominio de la lista...[3]\n");
55     printf("Eliminar un elemento al inicio de la lista.....[4]\n");
56     printf("Eliminar un elemento al final de la lista.....[5]\n");
57     printf("Eliminar un elemento en un punto intermedio dentro del dominio de la lista...[6]\n");
58     printf("Vaciar lista.....[7]\n");
59     printf("Concatenar listas.....[8]\n");
60     printf("Numeros repetidos en ambas listas.....[9]\n");
61     printf("Pasar un dato de una lista a otra.....[10]\n");
62     printf("Salir del programa.....[11]\n");
63
64     printf("\nPrimer lista:\n");
65     ImpresionLista(Lista);
66
67     printf("\nSegunda lista:\n");
68     ImpresionLista(SL);
69
70     if(nelista==1){
71         printf("\nEl valor eliminado de la primera lista fue: %d\n", eliminado);
72     }
73     else if(nelista==2){
74         printf("\nEl valor eliminado de la primera lista fue: %d\n", eliminado);
75     }
76     else{
77         printf("\n");
78     }
79
80     printf("\nPor favor escoja una opcion del menu:\n");
81     scanf("%d",&opcion);
-- }
```

En primera instancia podemos destacar la impresión de ambas listas, además de la impresión del dato eliminado que en este caso solo se mostrara cuando se elimine un dato de cualquier lista. Asimismo tenemos tres nuevas opciones las cuales trataran de cumplir con el objetivo de la implementación de operaciones entre dos listas. Para la mayoría de las operaciones básicas se presenta un cambio muy importante antes de entrar en la función.

```

80      printf("\nPor favor escoja una opcion del menu:\n");
81      scanf("%d",&opcion);
82
83      if(opcion==1){
84          printf("En cual lista gustas guardar tu dato?\n");
85          printf("Primera lista.....[1]\n");
86          printf("Segunda lista.....[2]\n");
87          scanf("%d",&nlista);
88          if(nlista==1){
89              printf("Introduzca el dato de su preferencia\n");
90              scanf("%d",&elemento);
91              InsertarInicio(Lista,elemento);
92          }
93          else if(nlista==2){
94              printf("Introduzca el dato de su preferencia\n");
95              scanf("%d",&elemento);
96              InsertarInicio(SL,elemento);
97          }
98          else
99              printf("\n Opcion invalida\n");
100      }

```

Como podemos observar a cambio de no realizar algún cambio notorio en cada una de las funciones que involucre cualquier operación básica de la lista, simplemente se optó por preguntar en que lista el usuario desea realizar el cambio, generando un control más simple con los elementos de cada lista.

Una vez explicadas las diferencias con el original, por parte de las operaciones básicas, ahora tomaremos como punto central las nuevas opciones que nos otorga el menú.

```

207      else if(opcion==8){
208          printf("En cual lista gustas guardar tu dato?\n");
209          printf("Primera lista.....[1]\n");
210          printf("Segunda lista.....[2]\n");
211          scanf("%d",&nlista);
212          if(nlista==1){
213              ConcatenarListas(Lista, SL);
214          }
215          else if(nlista==2){
216              ConcatenarListas(SL, Lista);
217          }
218          else
219              printf("\n Opcion invalida\n");
220          ConcatenarListas(Lista, SL);
221      }
222      else if(opcion==9){
223          BuscarComunes(Lista, SL);
224      }
225      else if(opcion==10){
226          printf("De que lista deseas cambiar el dato?\n");
227          printf("Primera lista.....[1]\n");
228          printf("Segunda lista.....[2]\n");
229          scanf("%d",&nlista);
230          if(nlista==1){
231              CambiarDato(Lista,SL);
232          }
233          else if(nlista==2){
234              CambiarDato(SL,List);
235          }
236          else
237              printf("\n Opcion invalida\n");

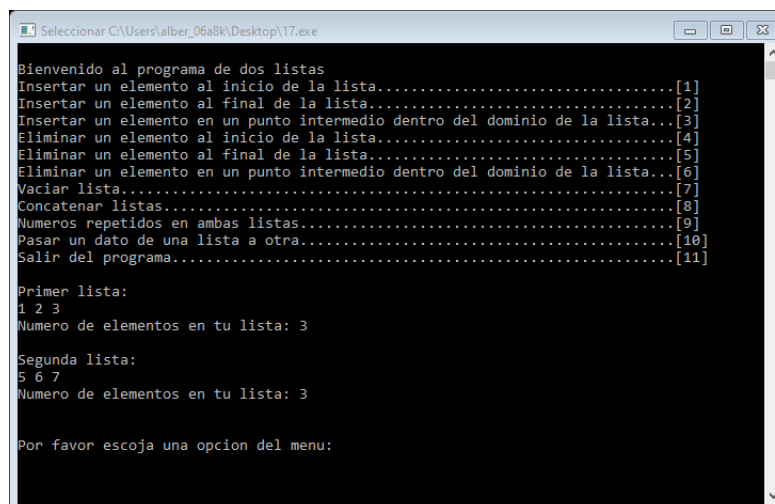
```

Por parte de "ConcatenarListas" esta al igual que las demás opciones del menú solicita la lista destino en la cual los valores de la otra lista serán unidos.

```
402 void ConcatenarListas(Nodo *plista, Nodo *slista){
403     int cont=0, rept=0;
404     if(plista->numelm==0 && slista->numelm==0){
405         printf("Ambas listas se encuentran vacías, no es posible realizar esta acción\n");
406         return;
407     }
408     else if(plista->numelm==0){
409         printf("La primera lista se encuentra vacía , por lo tanto, no es posible concatenar ambas listas\n");
410         return;
411     }
412     else if(slista->numelm==0){
413         printf("La segunda lista se encuentra vacía, por lo tanto, no es posible concatenar ambas listas\n");
414         return;
415     }
416     else
417     {
418         cont=plista->numelm;
419         do{
420             plista->dato[cont]=slista->dato[rept];
421             cont++;
422             rept++;
423         }while(rept!=slista->numelm);
424         plista->numelm=cont;
425         CrearOVaciarLista(slista);
426         return;
427     }
428 }
```

En primera instancia la función considera aquellos casos donde no es posible realizar la acción de concatenar, de manera general son 3 casos los cuales no permiten trabajar a esta función, tomando cada una como punto central el hecho de que alguna lista se encuentre vacía.

Posteriormente al no cumplirse los tres casos anteriores, procederemos a realizar la concatenación de ambas listas. Para esto se considera al apuntador "plista" como la lista destino, posteriormente al ser estáticas haremos uso de las condiciones de almacenamiento en cada una de las posiciones de ambas listas mediante variables enteras que estarán en constante aumento para asegurar la posición de los datos en la lista destino sin modificar ningún dato.



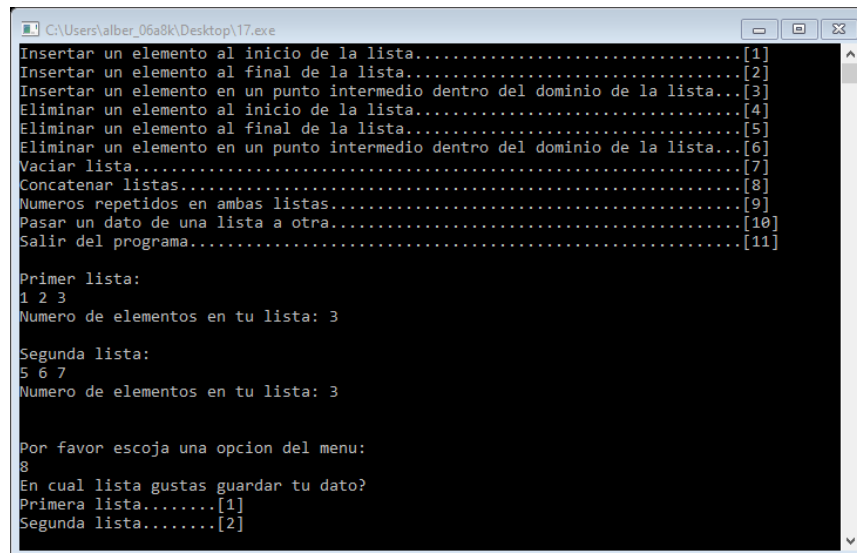
```
Seleccional C:\Users\alber_06a8\Desktop\17.exe
Bienvenido al programa de dos listas
Insertar un elemento al inicio de la lista.....[1]
Insertar un elemento al final de la lista.....[2]
Insertar un elemento en un punto intermedio dentro del dominio de la lista...[3]
Eliminar un elemento al inicio de la lista.....[4]
Eliminar un elemento al final de la lista.....[5]
Eliminar un elemento en un punto intermedio dentro del dominio de la lista...[6]
Vaciar lista.....[7]
Concatenar listas.....[8]
Numeros repetidos en ambas listas.....[9]
Pasar un dato de una lista a otra.....[10]
Salir del programa.....[11]

Primer lista:
1 2 3
Numero de elementos en tu lista: 3

Segunda lista:
5 6 7
Numero de elementos en tu lista: 3

Por favor escoja una opcion del menu:
```

Para demostrar la función de esta opción primero contaremos con dos listas que mínimo tendrán un elemento. Al seleccionar la opción de concatenar deberemos decidir cuál será la lista destino.



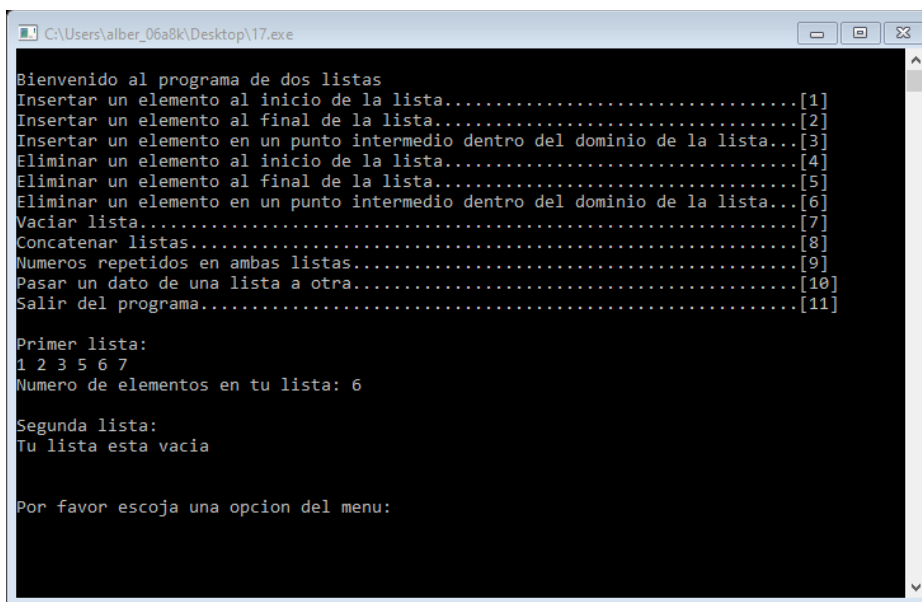
```
C:\Users\alber_06a8k\Desktop\17.exe
Insertar un elemento al inicio de la lista.....[1]
Insertar un elemento al final de la lista.....[2]
Insertar un elemento en un punto intermedio dentro del dominio de la lista...[3]
Eliminar un elemento al inicio de la lista.....[4]
Eliminar un elemento al final de la lista.....[5]
Eliminar un elemento en un punto intermedio dentro del dominio de la lista...[6]
Vaciar lista.....[7]
Concatenar listas.....[8]
Numeros repetidos en ambas listas.....[9]
Pasar un dato de una lista a otra.....[10]
Salir del programa.....[11]

Primer lista:
1 2 3
Numero de elementos en tu lista: 3

Segunda lista:
5 6 7
Numero de elementos en tu lista: 3

Por favor escoja una opcion del menu:
8
En cual lista gustas guardar tu dato?
Primera lista.....[1]
Segunda lista.....[2]
```

Para fines demostrativos decidimos que la lista destino será la primera lista del menú.



```
C:\Users\alber_06a8k\Desktop\17.exe
Bienvenido al programa de dos listas
Insertar un elemento al inicio de la lista.....[1]
Insertar un elemento al final de la lista.....[2]
Insertar un elemento en un punto intermedio dentro del dominio de la lista...[3]
Eliminar un elemento al inicio de la lista.....[4]
Eliminar un elemento al final de la lista.....[5]
Eliminar un elemento en un punto intermedio dentro del dominio de la lista...[6]
Vaciar lista.....[7]
Concatenar listas.....[8]
Numeros repetidos en ambas listas.....[9]
Pasar un dato de una lista a otra.....[10]
Salir del programa.....[11]

Primer lista:
1 2 3 5 6 7
Numero de elementos en tu lista: 6

Segunda lista:
Tu lista esta vacia

Por favor escoja una opcion del menu:
```

Como logramos observar la concatenación fue un éxito, dejando vacía la segunda lista para seguirla ocupando.

Ahora deberemos describir la opción "BuscarComunes" que como su nombre lo indica solo nos mostrara los números que se encuentran repetidos en ambas listas.

```
C:\Users\alber_06a8k\Desktop\17.exe

Bienvenido al programa de dos listas
Insertar un elemento al inicio de la lista.....[1]
Insertar un elemento al final de la lista.....[2]
Insertar un elemento en un punto intermedio dentro del dominio de la lista...[3]
Eliminar un elemento al inicio de la lista.....[4]
Eliminar un elemento al final de la lista.....[5]
Eliminar un elemento en un punto intermedio dentro del dominio de la lista...[6]
Vaciar lista.....[7]
Concatenar listas.....[8]
Numeros repetidos en ambas listas.....[9]
Pasar un dato de una lista a otra.....[10]
Salir del programa.....[11]

Primer lista:
4 5 6 7
Numero de elementos en tu lista: 4

Segunda lista:
6 7 8 9
Numero de elementos en tu lista: 4

Por favor escoja una opcion del menu:
9
```

```
440 void BuscarComunes(Nodo *plista, Nodo *slista){
441     int rpt=0, cont=0, grd=0, mostr=0;
442     TipoDato aux[MAX];
443     if(plista->numelm==0 && slista->numelm==0){
444         printf("Ambas listas se encuentran vacias, no es posible realizar esta accion\n");
445         return;
446     }
447     else if(plista->numelm==0){
448         printf("La primera lista se encuentra vacia , por lo tanto, no es posible concatenar ambas listas\n");
449         return;
450     }
451     else if(slista->numelm==0){
452         printf("La segunda lista se encuentra vacia, por lo tanto, no es posible concatenar ambas listas\n");
453         return;
454     }
455     else
456     do{
457         do{
458             if(plista->dato[rpt]==slista->dato[cont]){
459                 aux[grd]=plista->dato[rpt];
460                 grd++;
461                 cont++;
462                 mostr=rpt;
463                 mostr++;
464                 printf("\nEl numero %d de la posicion %d perteneciente a la primera lista\n",plista->dato[rpt],mostr);
465                 printf("\nSe repitio en la posicion %d de la segunda lista\n",cont);
466             }
467             else{
468                 cont++;
469             }
470         }while(cont!=slista->numelm);
```

Como podemos observar, primero se consideran aquellos casos donde sería inútil realizar dicha acción, al igual que la función de concatenar considera los mismos casos donde el punto central es el que una lista se encuentre vacía. Posteriormente mediante el uso de otros apuntadores a nodo llevaremos a cabo la comparación de cada valor presente en ambas listas. Presentando en donde se repitió durante algunos segundos para evitar el system("cls").

```

C:\Users\alber_06a8k\Desktop\17.exe
Numeros repetidos en ambas listas.....[9]
Pasar un dato de una lista a otra.....[10]
Salir del programa.....[11]

Primer lista:
4 5 6 7
Numero de elementos en tu lista: 4

Segunda lista:
6 7 8 9
Numero de elementos en tu lista: 4

Por favor escoja una opcion del menu:
9

El numero 6 de la posicion 3 perteneciente a la primera lista
Se repitio en la posicion 1 de la segunda lista

El numero 7 de la posicion 4 perteneciente a la primera lista
Se repitio en la posicion 2 de la segunda lista

Los numeros que se repitieron fueron los siguientes:
6 7

```

Por ultimo describiremos la opción de “CambiarDatos”, al igual que en “ConcatenarDato” primero deberemos seleccionar la lista destino.

```

else if(opcion==10){
    printf("De que lista deseas cambiar el dato?\n");
    printf("Primera lista.....[1]\n");
    printf("Segunda lista.....[2]\n");
    scanf("%d",&nlista);
    if(nlista==1){
        CambiarDato(Lista,SL);
    }
    else if(nlista==2){
        CambiarDato(SL,List);
    }
    else
        printf("\n Opcion invalida\n");
}

```

```

C:\Users\alber_06a8k\Desktop\17.exe
Insertar un elemento al final de la lista.....[2]
Insertar un elemento en un punto intermedio dentro del dominio de la lista...[3]
Eliminar un elemento al inicio de la lista.....[4]
Eliminar un elemento al final de la lista.....[5]
Eliminar un elemento en un punto intermedio dentro del dominio de la lista...[6]
Vaciar lista.....[7]
Concatenar listas.....[8]
Numeros repetidos en ambas listas.....[9]
Pasar un dato de una lista a otra.....[10]
Salir del programa.....[11]

Primer lista:
4 5 6 7
Numero de elementos en tu lista: 4

Segunda lista:
6 7 8 9
Numero de elementos en tu lista: 4

Por favor escoja una opcion del menu:
10
De que lista deseas cambiar el dato?
Primera lista.....[1]
Segunda lista.....[2]
1

```

Para casos prácticos del ejemplo decidiremos cambiar el dato 8 de la segunda lista, a la última posición de la primera lista.

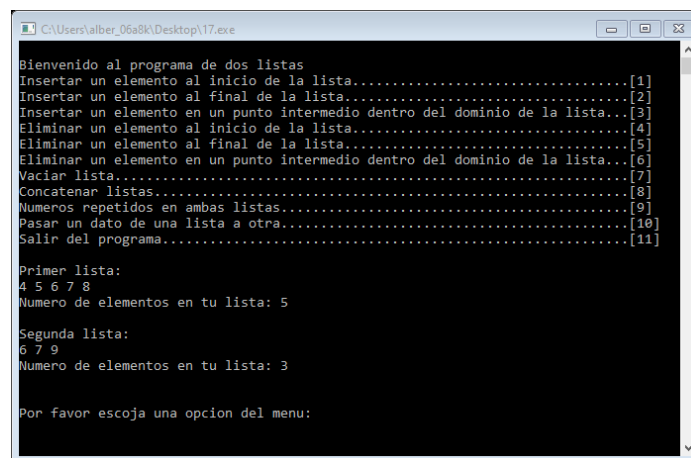
```

485 void CambiarDato(Nodo *plista, Nodo *slista){
486     int dop,op,posicion,postd;
487     TipoDato rspd;
488     if(plista->numelm==0 && slista->numelm==0){
489         printf("Ambas listas se encuentran vacias, no es posible realizar esta accion\n");
490         return;
491     }
492     else if(plista->numelm==0){
493         printf("La primera lista se encuentra vacia , por lo tanto, no es posible realizar esta accion\n");
494         return;
495     }
496     else
497     {
498         printf("\nQue posicion es la del dato deseas mover?\n");
499         printf("\nTu lista:\n");
500         ImpresionLista(plista);
501         printf("Principio.....[1]\n");
502         printf("Final.....[2]\n");
503         printf("Posicion intermedia.....[3]\n");
504         scanf("%d",&op);
505
506         printf("\nA que posicion de la otra lista deseas mover el dato?\n");
507         printf("\nLista destino:\n");
508         ImpresionLista(slista);
509         printf("Principio.....[1]\n");
510         printf("Final.....[2]\n");
511         printf("Posicion intermedia.....[3]\n");
512         scanf("%d",&dop);
513
514         if(op==1){
515             rspd=EliminarInicio(plista);
516             if(dop==1){
517                 InsertarInicio(slista,rspd);
518                 return;
519             }
520             else if(dop==2){
521                 InsertarFinal(slista,rspd);
522                 return;
523             }
524             else if(dop==3){
525                 printf("Introduzca la posicion de su preferencia de la lista destino\n");
526                 scanf("%d",&postd);
527                 InsertarCPI(slista,rspd,postd);
528                 return;
529             }
530             else{
531                 printf("\n Opcion invalida \n");
532                 return;
533             }
534         }
535         else if(op==2){
536             rspd=EliminarFinal(plista);
537             if(dop==1){
538                 InsertarInicio(slista,rspd);
539                 return;
540             }
541             else if(dop==2){
542                 InsertarFinal(slista,rspd);
543                 return;
544             }
545             else if(dop==3){
546                 printf("Introduzca la posicion de su preferencia de la lista destino\n");
547                 scanf("%d",&postd);
548                 InsertarCPI(slista,rspd,postd);
549                 return;
550             }
551             else{
552                 printf("\n Opcion invalida \n");
553                 return;
554             }
555         }
556         else if(op==3){
557             printf("Introduzca la posicion de su preferencia con respecto a los datos de la lista del dato que desea mover\n");
558             scanf("%d",&posicion);
559             rspd=EliminarCPI(plista,posicion);
560             if(dop==1){
561                 InsertarInicio(slista,rspd);
562                 return;
563             }
564             else if(dop==2){
565                 InsertarFinal(slista,rspd);
566                 return;
567             }
568             else if(dop==3){
569                 printf("Introduzca la posicion de su preferencia de la lista destino\n");
570                 scanf("%d",&postd);
571                 InsertarCPI(slista,rspd,postd);
572                 return;
573             }
574             else{
575                 printf("\n Opcion invalida \n");
576                 return;
577             }
578         }
579         else
580         {
581             printf("\n Opcion invalida \n");
582             return;
583         }
584     }
585 }

```


Una vez dentro de la función, se verifica que el valor exista en la lista donante, posteriormente se le pregunta al usuario donde prefiere colocar dicho elemento ya sea al final, principio o alguna posición intermedia de la lista destino. En base a la decisión tomada se comienza a verificar el caso presente de acuerdo a las posiciones del elemento donado y el espacio de la lista destino. Terminando con la aplicación de las operaciones básicas de este tipo de TAD lista.

Con lo que respecta al ejemplo nos despegara diversas preguntas para confirmar la posición del donante y el destino, obteniendo como resultado lo siguiente



```
C:\Users\alber_0688\Desktop\17.exe
Bienvenido al programa de dos listas
Insertar un elemento al inicio de la lista.....[1]
Insertar un elemento al final de la lista.....[2]
Insertar un elemento en un punto intermedio dentro del dominio de la lista...[3]
Eliminar un elemento al inicio de la lista.....[4]
Eliminar un elemento al final de la lista.....[5]
Eliminar un elemento en un punto intermedio dentro del dominio de la lista...[6]
Vaciar lista.....[7]
Concatenar listas.....[8]
Numeros repetidos en ambas listas.....[9]
Pasar un dato de una lista a otra.....[10]
Salir del programa.....[11]

Primer lista:
4 5 6 7 8
Numero de elementos en tu lista: 5

Segunda lista:
6 7 9
Numero de elementos en tu lista: 3

Por favor escoja una opcion del menu:
```

Operación de cambio:

```
Concatenar listas.....[8]
Numeros repetidos en ambas listas.....[9]
Pasar un dato de una lista a otra.....[10]
Salir del programa.....[11]

Primer lista:
4 5 6 7
Numero de elementos en tu lista: 4

Segunda lista:
6 7 8 9
Numero de elementos en tu lista: 4

Por favor escoja una opcion del menu:
10
De que lista deseas cambiar el dato?
Primera lista.....[1]
Segunda lista.....[2]
2
Que posicion es la del dato deseas mover?
Tu lista:
6 7 8 9
Numero de elementos en tu lista: 4
Principio.....[1]
Final.....[2]
Posicion intermedia.....[3]
3
A que posicion de la otra lista deseas mover el dato?
Lista destino:
4 5 6 7
Numero de elementos en tu lista: 4
Principio.....[1]
Final.....[2]
Posicion intermedia.....[3]
2
Introduzca la posicion de su preferencia con respecto a los datos de la lista del dato que desea mover
3
```

Programa 18

“Un aeropuerto internacional, desea controlar el flujo de pasajeros, y de aerolíneas que circulan por él. Diseñar un programa que dé soporte a las salidas de los aviones, mediante una lista doblemente enlazada cuya información sería la siguiente: Destino, compañía, hora de salida y pasajeros. Luego, y cuando se cumpla la hora de salida, es que se eliminarán los datos de la lista.”

Sabemos que una lista doblemente enlazada, es una lista lineal en la que cada elemento tiene dos enlaces, uno al nodo siguiente y otro al nodo anterior.



De igual forma que los anteriores programas por parte del encabezado podemos observar los archivos de cabecera y estructuras a utilizar a lo largo del programa. Destacando el nodo ya que presenta diversos cambios con respecto a los elementos que almacena debido a las características del problema. Asimismo para solucionar el problema del tiempo se decidió hacer una estructura donde se pueda modificar el horario del programa según las especificaciones de la demostración.

```
1  #include <string.h>
2  #include <stdio.h>
3  #include <conio.h>
4
5  typedef struct DPST{
6      char *destino;
7      char *company;
8      char *salida;
9      int pasajeros;
10     int posicion;
11     struct DPST *next;
12     struct DPST *before;
13 }Nodo;
14
15 typedef struct INL{
16     int nelementos;
17     struct DPST *PN;
18 }Indicador;
19
20 typedef struct agenda{
21     int horas;
22     int minutos;
23 }Tiempo;
```

Por parte de las variables definidas para el uso en la función main, podemos notar la definición del horario con el cual empezara el programa, además de las variables que hacen posible la introducción de cadenas de tipo caracteres para tener completa libertad en la definición de sus respectivos datos.

```
Indicador *Lista;
Lista=CrearLista(Lista);
Tiempo *actual;
actual=(Tiempo*)malloc(sizeof(Tiempo));
actual->horas=7;
actual->minutos=20;
int pasajeros, posicion, opcion, mh, mm;
Nodo *eliminado;
char *destino, *company, *salida;
destino=(char*)malloc(sizeof(char));
company=(char*)malloc(sizeof(char));
salida=(char*)malloc(sizeof(char));
```

```
27  Indicador* CrearLista(Indicador *lista);
28
29  Nodo* CrearNodo(char *fate,char *sociedad,char *exit,int clientes);
30
31  void ImprimirLista(Indicador *lista);
32
33  void ImprimirPosiciones(Indicador *lista);
34
35  void ImprimirNelementos(Indicador *lista);
36
37  void ImprimirHorario (Tiempo *ahora);
38
39  void ComparadorHorario(Indicador *lista, int horas, int minutos);
40
41  void InvertirCambioH(Indicador *lista, Tiempo *ahora, int mh, int mm);
42
43  void EliminarLista(Indicador *lista);
44
45  void InsertarInicio(Indicador *lista, char *fate, char *sociedad, char *exit, int clientes);
46
47  void InsertarFinal(Indicador *lista, char *fate, char *sociedad, char *exit, int clientes);
48
49  void InsertarCPI(Indicador *lista, char *fate, char *sociedad, char *exit, int clientes, int PP);
50
51  void EliminarInicio(Indicador *lista);
52
53  void EliminarFinal(Indicador *lista);
54
55  void EliminarCPI(Indicador *lista, int PP);
```

Por parte de los encabezados podemos destacar la aparición de las operaciones básicas con las cuales cuenta este tipo de TAD lista, además de aquellas que hacen uso del tiempo.

```

printf("Bienvenido al programa del aeropuerto\n");
printf("Opciones del programa\n");
printf("\nInsertar datos del vuelo al principio de la lista.....[1]\n");
printf("\nInsertar datos del vuelo en una posición intermedia.....[2]\n");
printf("\nInvertir.....[3]\n");
printf("\nInvertir.....[4]\n");
printf("\nSalir del programa.....[5]\n");

ComparadorHorario(Lista,actual->horas,actual->minutos);

printf("\nTu lista:\n");
ImprimirLista(Lista);
ImprimirNelementos(Lista);

printf("\nPor favor escoje una opcion del programa          Horario: ");
ImprimirHorario (actual);
scanf("%d",&opcion);
fflush(stdin);

```

Por parte del menú de opciones, debido a las características del problema no se cuenta con alguna opción que elimine los datos de un vuelo de forma manual. Lo interesante radica en la opción invertir que se describirá más adelante. Asimismo podemos notar más cambios como la presencia del horario y la función "ComparadorHorario" cuya función como el mismo nombre lo indica es la de comparar el horario presente y la hora de salida de algún vuelo.

Una vez que tocamos el tema describiremos la función "ComparadorHorario", lo primero que podemos notar es que necesita la lista y el horario dividido en horas y minutos.

```

259 void ComparadorHorario(Indicador *lista, int horas, int minutos){
260     Nodo *aux;
261     int lhrs, lmin, cont=1;
262     char *apy=(char*)malloc(sizeof(char));
263     if(lista->nelementos==0){
264         return;
265     }
266     else
267     aux=lista->PN;
268     do{
269         strcpy(apy,aux->salida);
270         lhrs=atoi(apy);
271         if(strlen(apy)==5){
272             lmin=atoi(apy+3);
273         }
274         else if(strlen(apy)==4){
275             lmin=atoi(apy+2);
276         }
277         else if(strlen(apy)>5 || strlen(apy)<4){
278             printf("El horario no esta bien escrito en el formato de 24 horas, por favor vuelvelo a intentar");
279             return;
280         }
281
282         if(horas==lhrs && minutos==lmin){
283             printf("\nEl vuelo con:\n");
284             printf("Destino:          %s\nCompañia:          %s\nHora de salida:");
285             printf("Acaba de salir\n");
286             sleep(3);
287             if(aux->posicion==1){
288                 aux=NULL;
289                 EliminarInicio(lista);
290                 aux=lista->PN;
291             }
292             else if(aux->posicion==lista->nelementos){
293                 EliminarFinal(lista);
294                 return;
295             }
296             else{
297                 aux=aux->next;
298                 EliminarCPI(lista, (aux->before)->posicion);
299             }
300         }
301         else{
302             aux=aux->next;
303         }
304     }while(aux!=NULL);
305     return;
306 }

```

Lo primero que nos interesa al analizar esta función es que no se encuentre vacía, posteriormente mediante el uso del apuntador a nodo "aux" apuntaremos al primer elemento de la lista ya que este actuará como el responsable de comparar los horarios de cada vuelo con el presente por parte del programa. Al entrar al bucle copiaremos la hora de salida en una variable auxiliar "apy" de la función para no modificar el texto original, se debe destacar que el horario de salida por parte del nodo se encuentra en formato de 24 horas, además que es una cadena de caracteres. Una vez copiada la cadena haremos uso de la función "atoi" el cual de forma general nos permite transformar una cadena de tipo carácter que cuente con números a datos de tipo entero, esto se hace ya que los valores del horario son de tipo entero, por ende el compararlos en su estado normal resulta en un error, pero hay un problema, la función "atoi" solo convierte los caracteres numéricos por ende al toparse con la expresión ":" para y solo guarda los valores obtenidos antes de dicha expresión, por lo tanto solo guardaría las horas. Por ende se utilizan dos variables de tipo entero auxiliares por parte de la función "ComparadorHorario" para guardar las

horas y los minutos por separado gracias a la función "atoi". Posteriormente comparamos las horas y minutos del horario por defecto y las del vuelo de salida. En el caso de que no sean iguales, la variable "aux" pasa al siguiente nodo hasta terminar la lista. Pero si resultan iguales se mostrara que vuelo salió además que la eliminación se hace de manera automática por medio de la comparación en base a la posición del nodo, llamando a las operaciones de eliminación básicas con las que cuenta el TAD lista doblemente enlazada.

```
512 void EliminarInicio(Indicador *lista){
513     Nodo *Nlib, *aux;
514     if(lista->nelementos==0){
515         printf("Tu lista esta vacia\n");
516         return;
517     }
518     else if(lista->nelementos==1){
519         free(lista->PN);
520         lista->PN=NULL;
521         lista->nelementos--;
522         printf("Eliminaste el ultimo elemento de tu lista\n");
523         return;
524     }
525     else
526     {
527         Nlib=lista->PN;
528         lista->PN=(lista->PN)->next;
529         (lista->PN)->before=NULL;
530         lista->nelementos--;
531         aux=lista->PN;
532         do{
533             aux->posicion--;
534             aux=aux->next;
535         }while(aux!=NULL);
536         free(Nlib);
537         return;
538     }
539 }
```

```
539 void EliminarFinal(Indicador *lista){
540     Nodo *Nlib, *aux;
541     if(lista->nelementos==0){
542         printf("Tu lista esta vacia\n");
543         return;
544     }
545     else if(lista->nelementos==1){
546         free(lista->PN);
547         lista->PN=NULL;
548         lista->nelementos--;
549         printf("Eliminaste el ultimo elemento de tu lista\n");
550         return;
551     }
552     else
553     {
554         Nlib=lista->PN;
555         do{
556             Nlib=Nlib->next;
557         }while(Nlib->posicion!=lista->nelementos);
558         (Nlib->before)->next=NULL;
559         lista->nelementos--;
560         free(Nlib);
561         return;
562     }
563 }
```

```

613 void EliminarCPI(Indicador *lista, int PP){
614     Nodo *Nlib, *aux;
615     if(lista->nelementos==0){
616         printf("Tu lista se encuentra vacia\n");
617         return;
618     }
619     else if(PP>=lista->nelementos){
620         printf("La posicion que solicito no es valida pruebe con la opcion de eliminar el final de la lista\n");
621         return;
622     }
623     else if(PP<=1){
624         printf("La posicion que solicito no es valida pruebe con la opcion de eliminar el inicio de la lista\n");
625         return;
626     }
627     else
628     Nlib=lista->PN;
629     do{
630         Nlib=Nlib->next;
631     }while(Nlib->posicion!=PP);
632     (Nlib->before)->next=Nlib->next;
633     (Nlib->next)->before=Nlib->before;
634     aux=Nlib->next;
635     do{
636         aux->posicion--;
637         aux=aux->next;
638     }while(aux!=NULL);
639     lista->nelementos--;
640     free(Nlib);
641     return;
642 }

```

Una vez que hablamos de las nuevas opciones del menú, ahora nos enfocaremos en las opciones que nos ofrece. Primero tenemos el insertar al inicio.

```

93     if(opcion==1){
94         printf("Por favor introduzca el destino del vuelo: ");
95         gets(destino);
96         printf("\nPor favor introduzca la compañía del vuelo: ");
97         gets(company);
98         printf("\nPor favor introduzca la hora de salida del vuelo en formato de 24 horas: ");
99         gets(salida);
100        fflush(stdin);
101        if(ComprobarHorario(salida)==1){
102            sleep(2);
103            system("cls");
104        }
105        else{
106            printf("\nPor favor introduzca el numero de pasajeros: ");
107            scanf("%d",&pasajeros);
108            fflush(stdin);
109            system("cls");
110            InsertarInicio(Lista,destino,company,salida,pasajeros);
111            InvernarCambioH(Lista, actual, 0, 5);
112        }
113    }

```

La única diferencia destacable es al momento de introducir los datos ya que debido a la naturaleza de estos deberemos hacer uso de otras funciones que nos permitan almacenar y verificar la correcta introducción de los datos. Claro ejemplo es la función "ComprobarHorario" la cual se encarga de verificar que la hora de salida del vuelo concuerde con los lineamientos del formato de 24 horas. Realizando una comparación similar a la función "ComparadorHorario" pero con diferentes lineamientos.

```

425 int ComprobarHorario(char *VHS){
426     int lhrs, lmin, cont=1;
427     char *apy=(char*)malloc(sizeof(char));
428     strcpy(apy,VHS);
429     lhrs=atoi(apy);
430     if(strlen(apy)==5){
431         lmin=atoi(apy+3);
432     }
433     else if(strlen(apy)==4){
434         lmin=atoi(apy+2);
435     }
436     else if(strlen(apy)>5 || strlen(apy)<4){
437         printf("El horario no esta bien escrito en el formato de 24 horas, por favor vuelvelo a intentar");
438         return 1;
439     }
440
441     if(lhrs>24){
442         printf("El horario no esta bien escrito en el formato de 24 horas, por favor vuelvelo a intentar");
443         return 1;
444     }
445     else if(lmin>59){
446         printf("El horario no esta bien escrito en el formato de 24 horas, por favor vuelvelo a intentar");
447         return 1;
448     }
449     else
450         return 0;
451 }

```

Posteriormente tenemos la llamada de la función básica "InsertarInicio" con el único cambio en los datos que solicita para su correcto funcionamiento.

```

455 void InsertarInicio(Indicador *lista, char *fate, char *sociedad, char *exit, int clientes){
456     Nodo *New;
457     if(lista->nelementos==0){
458         lista->PN=CrearNodo(fate, sociedad, exit, clientes);
459         lista->nelementos++;
460         (lista->PN)->next=NULL;
461         (lista->PN)->before=NULL;
462         (lista->PN)->posicion=lista->nelementos;
463         return;
464     }
465     else
466         New=CrearNodo(fate, sociedad, exit, clientes);
467     New->next=lista->PN;
468     New->before=NULL;
469     (lista->PN)->before=New;
470     New->posicion=1;
471     do{
472         (lista->PN)->posicion++;
473         lista->PN=(lista->PN)->next;
474     }while(lista->PN!=NULL);
475     lista->nelementos++;
476     lista->PN=New;
477     return;
478 }

```

Posteriormente y más importante tenemos la función "InvernarCambioH" la cual recibe la estructura del tiempo y dos valores de tiempo correspondiente a minutos y horas. De manera general esta función se encarga de dar la sensación de que el tiempo corre dentro del programa, ya que los otros elementos de tipo de entero solicitados, tienen como función ser el tiempo transcurrido después de cierta acción. Para lograr esto, mediante dos variables auxiliares "mm" (guarda minutos) y "mh" (guarda horas) de la mis función se igualan al valor de la suma del horario de la estructura y el de los valores introducidos por aparte. Una vez

realizada la suma y conversión para permanecer dentro de los lineamientos del formato de 24 horas se comparan las horas y minutos del horario de la estructura y los de las variables auxiliares, con el fin de brindar un aumento al tiempo. Con el fin de evitar un problema con los horarios de salida, después de cada aumento se llama a la función "ComparadorHorario".

```
328 void InvernarCambioH(Indicador *lista, Tiempo *ahora, int mh, int mm){
329     int aux,spt;
330     ImprimirHorario(ahora);
331     mm=mm+ahora->minutos;
332     ImprimirHorario(ahora);
333     if(mm>59){
334         aux=mm/59;
335         spt=59*aux;
336         mm=mm-spt;
337         mh=mh+aux;
338     }
339     mh=mh+ahora->horas;
340     ImprimirHorario(ahora);
341     if(mh>24){
342         aux=mh/24;
343         spt=24*aux;
344         mh=mh-spt;
345     }
346     ImprimirHorario(ahora);
347     if(ahora->horas==mh){
348         do{
349             ahora->minutos++;
350             if(ahora->minutos==60){
351                 ahora->horas++;
352                 ahora->minutos=0;
353                 ImprimirHorario(ahora);
354                 ComparadorHorario(lista,ahora->horas,ahora->minutos);
355                 system("cls");
356             }
357         } else if(ahora->horas==25){
358             ahora->horas=1;
359             ahora->minutos=0;
360             ImprimirHorario(ahora);
361             ComparadorHorario(lista,ahora->horas,ahora->minutos);
362             system("cls");
363         }
364         else{
365             ImprimirHorario(ahora);
366             ComparadorHorario(lista,ahora->horas,ahora->minutos);
367             system("cls");
368         }
369     }while(ahora->minutos!=mm);
370     return;
371 }
372 else
```

```

372 else
373 do{
374     ahora->minutos++;
375     if(ahora->minutos==60){
376         ahora->horas++;
377         ahora->minutos=0;
378         ImprimirHorario(ahora);
379         ComparadorHorario(lista,ahora->horas,ahora->minutos);
380         system("cls");
381     }
382     else if(ahora->horas==25){
383         ahora->horas=1;
384         ahora->minutos=0;
385         ImprimirHorario(ahora);
386         ComparadorHorario(lista,ahora->horas,ahora->minutos);
387         system("cls");
388     }
389     else{
390         ImprimirHorario(ahora);
391         ComparadorHorario(lista,ahora->horas,ahora->minutos);
392         system("cls");
393     }
394 }while(ahora->horas!=mh);
395
396 do{
397     ahora->minutos++;
398     if(ahora->minutos==60){
399         ahora->horas++;
400         ahora->minutos=0;
401         ImprimirHorario(ahora);
402         ComparadorHorario(lista,ahora->horas,ahora->minutos);
403         system("cls");
404     }
405     else if(ahora->horas==25){
406         ahora->horas=1;
407         ahora->minutos=0;
408         ImprimirHorario(ahora);
409         ComparadorHorario(lista,ahora->horas,ahora->minutos);
410         system("cls");
411     }
412     else{
413         ImprimirHorario(ahora);
414         ComparadorHorario(lista,ahora->horas,ahora->minutos);
415         system("cls");
416     }
417 }while(ahora->minutos!=mm);
418 return;
419 }

```

Ahora describiremos la función básica "InsertarFinal" con el único cambio en los datos que solicita para su correcto funcionamiento de igual forma que la opción "InsertarInicio".

```

else if(opcion==2){
    printf("Por favor introduzca el destino del vuelo: ");
    gets(destino);
    printf("\nPor favor introduzca la compañía del vuelo: ");
    gets(company);
    printf("\nPor favor introduzca la hora de salida del vuelo en formato de 24 horas: ");
    gets(salida);
    fflush(stdin);
    if(ComprobarHorario(salida)==1){
        sleep(2);
        system("cls");
    }
    else{
        printf("\nPor favor introduzca el numero de pasajeros: ");
        scanf("%d",&pasajeros);
        fflush(stdin);
        system("cls");
        InsertarFinal(Lista,destino,company,salida,pasajeros);
        InvertirCambioH(Lista, actual, 0, 5);
    }
}

```

Llamada de la función:

```

476 void InsertarFinal(Indicador *lista, char *fate, char *sociedad, char *exit, int clientes){
477     Nodo *New, *aux;
478     if(lista->nelementos==0){
479         lista->PN=CrearNodo(fate, sociedad, exit, clientes);
480         lista->nelementos++;
481         (lista->PN)->next=NULL;
482         (lista->PN)->before=NULL;
483         (lista->PN)->posicion=lista->nelementos;
484         return;
485     }
486     else if(lista->nelementos==1){
487         New=CrearNodo(fate, sociedad, exit, clientes);
488         New->before=lista->PN;
489         New->next=NULL;
490         (lista->PN)->next=New;
491         lista->nelementos++;
492         New->posicion=lista->nelementos;
493         return;
494     }
495     New=CrearNodo(fate, sociedad, exit, clientes);
496     aux=lista->PN;
497     do{
498         aux=aux->next;
499     }while(aux->posicion!=lista->nelementos);
500     New->before=aux;
501     New->next=NULL;
502     (New->before)->next=New;
503     lista->nelementos++;
504     New->posicion=lista->nelementos;
505     return;
506 }

```

Después tenemos la función básica "InsertarCPI" con el único cambio en los datos que solicita para su correcto funcionamiento de igual forma que la opción "InsertarInicio".

```

else if(opcion==3){
    printf("Por favor introduzca el destino del vuelo: ");
    gets(destino);
    printf("\nPor favor introduzca la compañía del vuelo: ");
    gets(company);
    printf("\nPor favor introduzca la hora de salida del vuelo en formato de 24 horas: ");
    gets(salida);
    fflush(stdin);
    if(ComprobarHorario(salida)==1){
        sleep(2);
        system("cls");
    }
    else{
        printf("\nPor favor introduzca el numero de pasajeros: ");
        scanf("%d",&pasajeros);
        printf("\nPor favor introduzca la posicion de su gusto: ");
        scanf("%d",&posicion);
        fflush(stdin);
        system("cls");
        InsertarCPI(Lista,destino,company,salida,pasajeros,posicion);
        InvernarCambioH(Lista, actual, 0, 5);
    }
}

```

Llamada de la función

```

561 void InsertarCPI(Indicador *lista, char *fate, char *sociedad, char *exit, int clientes, int PP){
562     Nodo *New, *aux;
563     if(lista->nelementos==0){
564         printf("Tu lista se encuentra vacia\n");
565         return;
566     }
567     else if(PP>=lista->nelementos){
568         printf("La posicion que solicito no es valida pruebe con la opcion de insertar al final de la lista\n");
569         return;
570     }
571     else if(PP<1){
572         printf("La posicion que solicito no es valida pruebe con la opcion de insertar al inicio de la lista\n");
573         return;
574     }
575     else if(PP==1){
576         New=CrearNodo(fate,sociedad,exit,clientes);
577         aux=(lista->PN)->next;
578         New->posicion=PP;
579         New->before=lista->PN;
580         New->next=aux;
581         (lista->PN)->next=New;
582         aux->before=New;
583         do{
584             New->posicion++;
585             New=New->next;
586         }while(New!=NULL);
587         lista->nelementos++;
588         return;
589     }
590     else
591     New=CrearNodo(fate,sociedad,exit,clientes);
592     aux=lista->PN;
593     do{
594         aux=aux->next;
595     }while(aux->posicion!=PP);
596     New->posicion=PP;
597     New->before=aux;
598     New->next=aux->next;
599     (aux->next)->before=New;
600     aux->next=New;
601     do{
602         New->posicion++;
603         New=New->next;
604     }while(New!=NULL);
605     lista->nelementos++;
606     return;
607 }

```

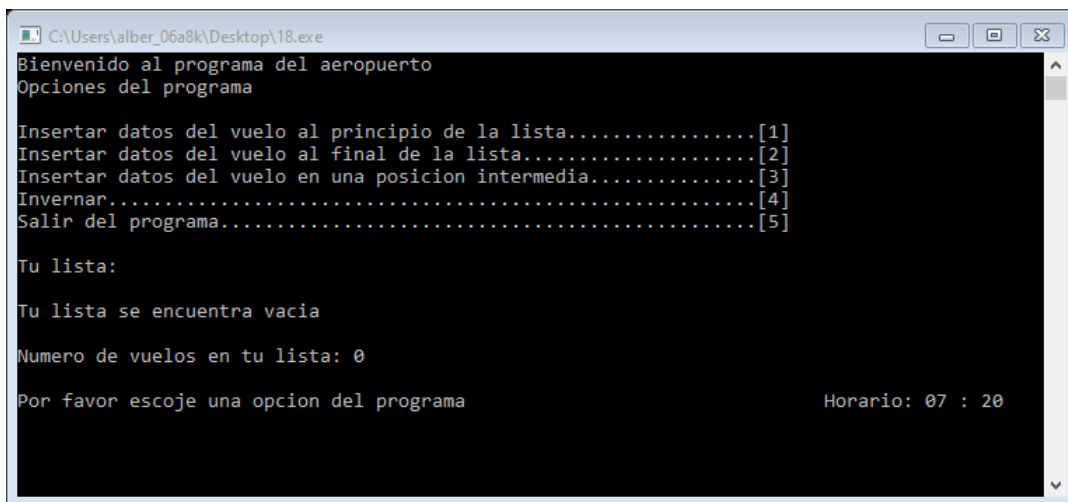
Por ultimo describiremos la opción "Invernar", como se comentó antes al llamar a la función "InvernarCambioH", este nos solicitara dos datos extras de tipo entero, al tener problemas con presentar el horario de forma

automática, como alternativa solo vote por hacer una demostración de la función "ComparadorHorario".

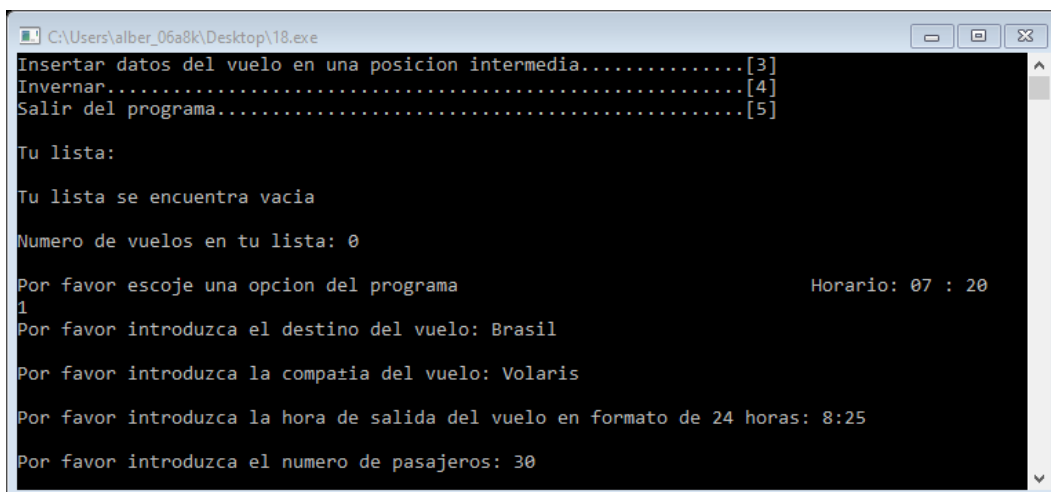
```
158 else if(opcion==4){
159     printf("Cuantas horas deseas que suspender el programa\n");
160     scanf("%d",&mh);
161     printf("Cuantos minutos deseas que suspender el programa\n");
162     scanf("%d",&mm);
163     if(mh>24 || mm>59){
164         printf("Valor invalido\n");
165     }
166     else{
167         InvernarCambioH(Lista, actual,mh,mm);
168         //system("cls");
169     }
170 }
```

Demostración de cada opción:

Menú

A screenshot of a Windows command prompt window titled "C:\Users\alber_06a8k\Desktop\18.exe". The program displays a welcome message "Bienvenido al programa del aeropuerto" and a menu titled "Opciones del programa". The menu lists five options: "Insertar datos del vuelo al principio de la lista.....[1]", "Insertar datos del vuelo al final de la lista.....[2]", "Insertar datos del vuelo en una posicion intermedia.....[3]", "Invernar.....[4]", and "Salir del programa.....[5]". Below the menu, it says "Tu lista:" followed by "Tu lista se encuentra vacia" and "Numero de vuelos en tu lista: 0". At the bottom, it prompts "Por favor escoje una opcion del programa" and shows the current time "Horario: 07 : 20".

Insertar Inicio

A screenshot of the same Windows command prompt window. The user has selected option 1, and the program prompts for flight details. It shows "Tu lista:" followed by "Tu lista se encuentra vacia" and "Numero de vuelos en tu lista: 0". The prompts are: "Por favor escoje una opcion del programa" (with '1' entered), "Por favor introduzca el destino del vuelo: Brasil", "Por favor introduzca la compaia del vuelo: Volaris", "Por favor introduzca la hora de salida del vuelo en formato de 24 horas: 8:25", and "Por favor introduzca el numero de pasajeros: 30". The time "Horario: 07 : 20" is still displayed.

```
C:\Users\alber_06a8k\Desktop\18.exe

Insertar datos del vuelo al principio de la lista.....[1]
Insertar datos del vuelo al final de la lista.....[2]
Insertar datos del vuelo en una posicion intermedia.....[3]
Invertir.....[4]
Salir del programa.....[5]

Tu lista:
Vuelo 1
Destino:          Brasil
Compania:         Volaris
Horario de salida: 8:25
Numero de pasajeros: 30

Numero de vuelos en tu lista: 1

Por favor escoje una opcion del programa                                Horario: 07 : 25
```

Insertar Final

```
C:\Users\alber_06a8k\Desktop\18.exe

Tu lista:
Vuelo 1
Destino:          Brasil
Compania:         Volaris
Horario de salida: 8:25
Numero de pasajeros: 30

Numero de vuelos en tu lista: 1

Por favor escoje una opcion del programa                                Horario: 07 : 25
2
Por favor introduzca el destino del vuelo: Japon

Por favor introduzca la compania del vuelo: Interjet

Por favor introduzca la hora de salida del vuelo en formato de 24 horas: 10:33

Por favor introduzca el numero de pasajeros: 28
```

```
C:\Users\alber_06a8k\Desktop\18.exe

Salir del programa.....[5]

Tu lista:
Vuelo 1
Destino:          Brasil
Compania:         Volaris
Horario de salida: 8:25
Numero de pasajeros: 30
Vuelo 2
Destino:          Japon
Compania:         Interjet
Horario de salida: 10:33
Numero de pasajeros: 28

Numero de vuelos en tu lista: 2

Por favor escoje una opcion del programa                                Horario: 07 : 30
```

Insertar en cualquier posición intermedia:

```
C:\Users\alber_06a8k\Desktop\18.exe
Destino: Japon
Compania: Interjet
Horario de salida: 10:33
Numero de pasajeros: 28

Numero de vuelos en tu lista: 2

Por favor escoje una opcion del programa Horario: 07 : 30
3
Por favor introduzca el destino del vuelo: Canada
Por favor introduzca la compaia del vuelo: Viva Aerobus
Por favor introduzca la hora de salida del vuelo en formato de 24 horas: 7:50
Por favor introduzca el numero de pasajeros: 21
Por favor introduzca la posicion de su gusto: 1
```

```
C:\Users\alber_06a8k\Desktop\18.exe
Bienvenido al programa del aeropuerto
Opciones del programa

Insertar datos del vuelo al principio de la lista.....[1]
Insertar datos del vuelo al final de la lista.....[2]
Insertar datos del vuelo en una posicion intermedia.....[3]
Invernar.....[4]
Salir del programa.....[5]

Tu lista:
Vuelo 1
Destino: Brasil
Compania: Volaris
Horario de salida: 8:25
Numero de pasajeros: 30
Vuelo 2
Destino: Canada
Compania: Viva Aerobus
Horario de salida: 7:50
Numero de pasajeros: 21
Vuelo 3
Destino: Japon
Compania: Interjet
Horario de salida: 10:33
Numero de pasajeros: 28

Numero de vuelos en tu lista: 3

Por favor escoje una opcion del programa Horario: 07 : 35
```

Invernar:

```
C:\Users\alber_06a8k\Desktop\18.exe
Bienvenido al programa del aeropuerto
Opciones del programa

Insertar datos del vuelo al principio de la lista.....[1]
Insertar datos del vuelo al final de la lista.....[2]
Insertar datos del vuelo en una posicion intermedia.....[3]
Invernar.....[4]
Salir del programa.....[5]

Tu lista:
Vuelo 1
Destino:          Brasil
Compania:         Volaris
Horario de salida: 8:25
Numero de pasajeros: 30
Vuelo 2
Destino:          Canada
Compania:         Viva Aerobus
Horario de salida: 7:50
Numero de pasajeros: 21
Vuelo 3
Destino:          Japon
Compania:         Interjet
Horario de salida: 10:33
Numero de pasajeros: 28

Numero de vuelos en tu lista: 3

Por favor escoje una opcion del programa                      Horario: 07 : 35
4
Cuantas horas deseas que suspender el programa
3
Cuantos minutos deseas que suspender el programa
23
```

```
Seleccionar C:\Users\alber_06a8k\Desktop\18.exe
10 : 33

El vuelo con:
Destino:          Japon
Compania:         Interjet
Hora de salida:   10:33
Num.Pasajeros: 28
Acaba de salir

█
```

```
C:\Users\alber_06a8k\Desktop\18.exe
Bienvenido al programa del aeropuerto
Opciones del programa

Insertar datos del vuelo al principio de la lista.....[1]
Insertar datos del vuelo al final de la lista.....[2]
Insertar datos del vuelo en una posicion intermedia.....[3]
Invernar.....[4]
Salir del programa.....[5]

Tu lista:

Tu lista se encuentra vacia

Numero de vuelos en tu lista: 0

Por favor escoje una opcion del programa                      Horario: 10 : 58
```