



TECNOLOGICO NACIONAL DE MEXICO CAMPUS QUERETARO.

De la Presilla Vega Víctor Hugo, 10141028

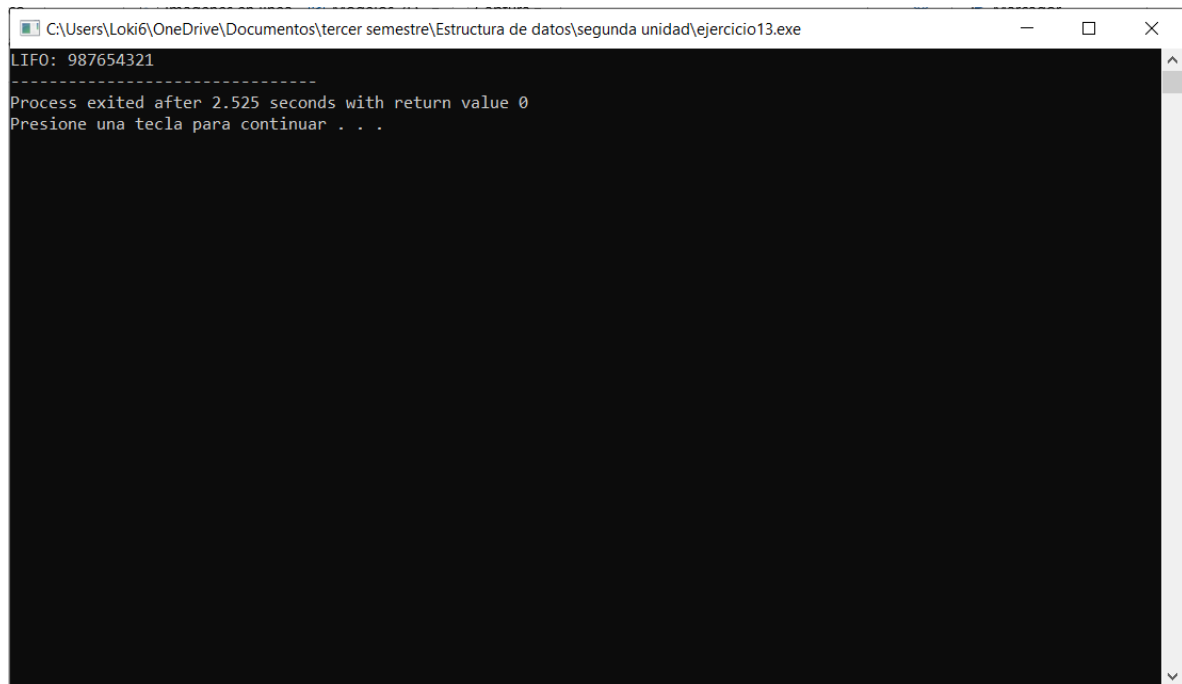
Facultad de ingeniería.

Carrera sistemas computacionales.

“Prácticas unidad 2”

Estructura de datos

```
1  /*pracrica 13*/
2  #include<stdio.h>
3  #include<stdlib.h>
4  int main()
5  {
6      static int pila[10]={0,1,2,3,4,5,6,7,8,9};
7      int i=0;
8      printf("LIFO: ");
9      for(i=9;i>0;i--)
10     {
11         printf("%d",pila[i]);
12     }
13     return 0;
14 }
```



```
C:\Users\Loki6\OneDrive\Documentos\tercer semestre\Estructura de datos\segunda unidad\ejercicio13.exe
LIFO: 987654321
-----
Process exited after 2.525 seconds with return value 0
Presione una tecla para continuar . . .
```

```

1  /*practica 17*/
2  #include<stdio.h>
3  #include<stdlib.h>
4  int e[10]={1,2,3,4,5,6,7,8,9,0};
5  int s[10]={11,22,33,44,55,66,77,88,99,00};
6  int main()
7  {
8      int *ee,*ss, dato=0,x=0;
9      ee=&e[0];
10     ss=&s[0];
11     for(x=0;x<10;++x)
12     {
13         do
14         {
15             printf("Valor de entrada: ");
16             scanf("%d",&dato);
17         }
18         while(dato!=*(ee+x));
19         printf("\nSalida=%d\n",*(ss+x));
20     }
21     system("PAUSE");
22     return 0;
23 }
24
25

```

```

C:\Users\Loki6\OneDrive\Documentos\tercer semestre\Estructura de datos\segunda unidad\ejercicio17.exe
Valor de entrada: 1
Salida=11
Valor de entrada:

```

```

1  /*practica 18*/
2  #include<stdio.h>
3  #include<stdlib.h>
4  int n[20]={3,4,5,1,2,8,9,6,7,8,12,10,11,2,1,0,4,3,2,15};
5  main()
6  {
7      int i,j,k=0;
8      int *po; po=&n[0];
9      printf("Lista original: ");
10     for(i=0;i<20;++i) printf("%d ", *(po+i));
11     for(i=0;i<19;++i)
12     {
13         for(j=i+1;j<20;++j)
14             if(*(po+i)>*(po+j))
15             {
16                 k=*(po+i); *(po+i)=*(po+j); *(po+j)=k;
17             }
18     }
19     printf("\nLista ordenada: ");
20     for(i=0;i<20;++i) printf("%d ", *(po+i));
21     system("PAUSE");
22     return 0;
23 }
24

```

```

C:\Users\Loki6\OneDrive\Documentos\tercer semestre\Estructura de datos\segunda unidad\ejercicio18.exe
Lista original: 3 4 5 1 2 8 9 6 7 8 12 10 11 2 1 0 4 3 2 15
Lista ordenada: 0 1 1 2 2 2 3 3 4 4 5 6 7 8 8 9 10 11 12 15 Presione una tecla para continuar . . .

```

```

2  /*pila simple*/
3  #include<iostream>
4  #include<stdlib.h>
5  using namespace std;
6  struct Nodo{
7      int dato;
8      Nodo *siguiente;
9  };
10 void agregarPila(Nodo *&, int);
11 void sacarPila(Nodo *&pila, int &);
12
13 int main(){
14     Nodo *pila= NULL;
15     int i,dato,c;
16
17     cout<<"¿Cuantos elementos hay?"<<endl; cin>>c;
18     for(i=0;i<c;i++){
19         cout <<"introduce un numero: "<<endl;
20         cin>>dato;
21         agregarPila(pila, dato);
22     }
23     cout <<"-----\nla pila esta llena\n-----"<<endl;
24
25     cout<<"\nSacando elementos de la pila: ";
26     while(pila != NULL){
27         sacarPila(pila,dato);
28         if(pila != NULL){
29             cout<<dato<<" , ";
30         }else{
31             cout<<dato<<" . ";
32         }
33     }
34     return 0;

```

```

C:\Users\Loki6\OneDrive\Documentos\tercer semestre\Estructura de datos\segunda unidad\Pila simple.exe
¿Cuantos elementos hay?
4
introduce un numero:
8
El elemento 8 se inserto a la pila
introduce un numero:
6
El elemento 6 se inserto a la pila
introduce un numero:
1
El elemento 1 se inserto a la pila
introduce un numero:
2
El elemento 2 se inserto a la pila
-----
La pila esta llena
-----

Sacando elementos de la pila: 2, 1, 6, 8.
-----
Process exited after 30.78 seconds with return value 0
Presione una tecla para continuar . . .

```

```

1  /*Pila con clases*/
2  #include<iostream>
3  using namespace std;
4
5  class Pila
6  {
7      private:
8          class Nodo
9          {
10             public:
11                 int info;
12                 Nodo *sig;
13             };
14
15             Nodo *raiz;
16         public:
17             Pila();
18             ~Pila();
19             void insertar (int x);
20             int extraer();
21             void imprimir();
22         };
23
24     Pila::Pila()
25     {
26         raiz = NULL;
27     }
28     void Pila::insertar(int x)
29     {
30         Nodo *nuevo;
31         nuevo = new Nodo();
32         nuevo->info=x;
33         if(raiz==NULL)
34         {
35             raiz = nuevo;
36             nuevo->sig=NULL;
37         }

```

```

38         else
39         {
40             nuevo->sig=raiz;
41             raiz = nuevo;
42         }
43     }
44     void Pila::imprimir()
45     {
46         Nodo *reco = raiz;
47         cout<< "Estos son todos los elemntos de la pila "<<endl;
48         while(reco != NULL)
49         {
50             cout<< reco->info<<",";
51             reco = reco->sig;
52         }
53         cout<<"\n";
54     }
55     int Pila::extraer()
56     {
57         if(raiz != NULL)
58         {
59             int informacion = raiz->info;
60             Nodo *bor = raiz;
61             raiz =raiz->sig;
62             delete bor;
63             return informacion;
64         }
65         else
66         {
67             return -1;
68         }
69     }
70     Pila::~~Pila()
71     {
72         Nodo *reco = raiz;
73         Nodo *bor;

```

```

74     while(reco != NULL)
75     {
76         bor =reco;
77         reco =reco->sig;
78         delete bor;
79     }
80 }
81 int main()
82 {
83     Pila *pila1;
84     pila1= new Pila();
85     int n;
86     char a;
87     while(true)
88     {
89         cout<<"Agrega elemento: "<<endl;
90         cin>>n;
91         pila1->insertar(n);
92         cout<<"Agregar otro elemento? Si=s; No=n"<<endl;
93         cin>>a;
94         if(a=='n')
95             goto ya;
96     }
97     ya:
100    pila1->imprimir();
101    cout<<"Extraer de la pila el elemento: "<<pila1->extraer()<<"\n";
102    pila1->imprimir();
103    delete pila1;
104    return 0;
105 }

```

```

C:\Users\Loki6\OneDrive\Documentos\tercer semestre\Estructura de datos\segunda unidad\Pila clases.exe
Agrega elemento:
1
Agregar otro elemento? Si=s; No=n
s
Agrega elemento:
4
Agregar otro elemento? Si=s; No=n
s
Agrega elemento:
8
Agregar otro elemento? Si=s; No=n
n
Estos son todos los elemntos de la pila
8,4,1,
Extraer de la pila el elemento: 8
Estos son todos los elemntos de la pila
4,1,
-----
Process exited after 50.29 seconds with return value 0
Presione una tecla para continuar . . .

```



```

1  /*cola simple*/
2  #include<iostream>
3  #include<stdlib.h>
4  using namespace std;
5  struct Nodo{
6      int dato;
7      Nodo *siguiente;
8  };
9  void insertarCola(Nodo *&,Nodo *&,int);
10 void suprimirCola(Nodo *&,Nodo *&,int &);
11 bool cola_vacia(Nodo *);
12 int main()
13 {
14     Nodo *frente = NULL;
15     Nodo*fin = NULL;
16     int dato;
17     char a;
18     while(true)
19     {
20         cout<<"Agrega elemento: "<<endl;
21         cin>>dato;
22         insertarCola(frente,fin,dato);
23         cout<<"Agregar otro elemento? Si=s; No=n"<<endl;
24         cin>>a;
25         if(a=='n')
26         {
27             goto ya;
28         }
29     }
30     ya:
31     cout<<"\nQuitando los datos de la cola"<<endl;
32     while(frente != NULL)
33     {
34         suprimirCola(frente,fin,dato);
35         if(frente != NULL)
36         {
37             cout<<dato<<" ";

```

```

38     }
39     else
40     {
41         cout<<dato<<". ";
42     }
43 }
44 return 0;
45 }
46 void insertarCola(Nodo *&frente, Nodo *&fin, int n)
47 {
48     Nodo *nuevo_nodo = new Nodo();
49     nuevo_nodo->dato = n;
50     nuevo_nodo->siguiente = NULL;
51     if(cola_vacia(frente))
52     {
53         frente= nuevo_nodo;
54     }
55     else
56     {
57         fin->siguiente=nuevo_nodo;
58     }
59     fin = nuevo_nodo;
60     cout<<"\tAgregaste un elemento"<<endl;
61 }
62 void suprimirCola(Nodo *&frente, Nodo *&fin, int &n)
63 {
64     n = frente->dato;
65     Nodo *aux =frente;
66     if(frente == fin)
67     {
68         frente = NULL;
69         fin = NULL;
70     }
71     else
72     {
73         frente = frente->siguiente;

```

```

38     }
39     else
40     {
41         cout<<dato<<". ";
42     }
43 }
44 return 0;
45 }
46 void insertarCola(Nodo *&frente, Nodo *&fin, int n)
47 {
48     Nodo *nuevo_nodo = new Nodo();
49     nuevo_nodo->dato = n;
50     nuevo_nodo->siguiente = NULL;
51     if(cola_vacia(frente))
52     {
53         frente = nuevo_nodo;
54     }
55     else
56     {
57         fin->siguiente = nuevo_nodo;
58     }
59     fin = nuevo_nodo;
60     cout<<"\tAgregaste un elemento"<<endl;
61 }
62 void suprimirCola(Nodo *&frente, Nodo *&fin, int &n)
63 {
64     n = frente->dato;
65     Nodo *aux = frente;
66     if(frente == fin)
67     {
68         frente = NULL;
69         fin = NULL;
70     }
71     else
72     {
73         frente = frente->siguiente;

```

```

74     }
75     delete aux;
76 }
77 bool cola_vacia(Nodo *frente)
78 {
79     if(frente == NULL)
80     {
81         return true;
82     }
83     else
84     {
85         return false;
86     }
87 }

```

C:\Users\Loki6\OneDrive\Documentos\tercer semestre\Estructura de datos\segunda unidad\colasimple.exe

```

4
Agregaste un elemento
Agregar otro elemento? Si=s; No=n
s
Agrega elemento:
8
Agregaste un elemento
Agregar otro elemento? Si=s; No=n
s
Agrega elemento:
7
Agregaste un elemento
Agregar otro elemento? Si=s; No=n
s
Agrega elemento:
7
Agregaste un elemento
Agregar otro elemento? Si=s; No=n
s
Agrega elemento:
0
Agregaste un elemento
Agregar otro elemento? Si=s; No=n
n

Quitando los datos de la cola
4, 8, 7, 7, 0.
-----
Process exited after 47.45 seconds with return value 0
Presione una tecla para continuar . . .

```

```

1 //cola circular
2 #include<stdlib.h>
3 #include<iostream>
4 #include <Stdio.h>
5 void visualizar(float cola[])
6 {
7     int i;
8     for(i=0;i<3;i++)
9     {
10         printf("[%f]",cola[i]);
11     }
12 }
13 int main()
14 {
15     float cola[3];
16     int frente=0;
17     int atras=0;
18     int band=0;
19     int opc;
20     float dato;
21     do{
22         printf("\n\n%c%c%c%c%c%c menu: %c%c%c%c%c%c",178,178,178,178,178,178,178,178,178,178,178,178,178);
23         printf("\n 1.insertar:");
24         printf("\n 2.Eliminar:");
25         printf("\n 3.mostrar: ");
26         printf("\n 4.Salir:");
27         printf("\n %cque quieres hacer?",168);
28         scanf("%d",&opc);
29         switch(opc)
30         {
31             case 1:
32                 if(frente==atras && band==1){printf("cola llena....\n");}
33             else
34             {
35                 printf("\n teclea un dato:");
36                 scanf("%f",&dato);
37                 cola[atras]=dato;

```

```

38         printf("dato insertado...%f",cola[atras]);
39         atras=(atras+1)%3;
40         band=1;
41     }
42     break;
43     case 2:
44         if(frente==atras && band==0){printf("cola vacia.....\n");}
45     else{
46         dato=cola[frente];
47         printf("dato eliminado...%f",dato);
48         band=0;
49         frente=(frente+1)%3;
50     }
51     break;
52     case 3:
53         visualizar(cola);
54         break;
55     default:
56         printf("\n\nesa opcion no esta disponible");
57 }
58 }
59 }
60 }
61 }
62 while(opc!=4);
63 system("pause");
64 return 0;
65 }
66 }

```

```
C:\Users\Loki6\OneDrive\Documentos\tercer semestre\Estructura de datos\segunda unidad\cola circular.exe
dato insertado...8.000000

menu:
1.insertar:
2.Eliminar:
3.mostrar:
4.Salir:
¿que quieres hacer?1

teclea un dato:4
dato insertado...4.000000

menu:
1.insertar:
2.Eliminar:
3.mostrar:
4.Salir:
¿que quieres hacer?3
[7.000000][8.000000][4.000000]

menu:
1.insertar:
2.Eliminar:
3.mostrar:
4.Salir:
¿que quieres hacer?
```

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <locale.h>
4  struct INFO{
5      int num;
6  };
7  struct NODO{
8      struct INFO elemento;
9      struct NODO *p_anterior;
10     struct NODO *p_siguiente;
11 };
12 struct BICOLA{
13     int nodos;
14     struct NODO *primero;
15     struct NODO *ultimo;
16 };
17 void inicializarBicola( struct BICOLA **bicola ){
18     struct BICOLA *temp = (struct BICOLA *) malloc(sizeof(struct BICOLA));
19     temp->nodos = 0;
20     temp->primero = NULL;
21     temp->ultimo = NULL;
22     (*bicola) = temp;
23 };
24 void insertIzqBicola( struct BICOLA **bicola, int dato ){
25
26     struct NODO *temp = (struct NODO *) malloc(sizeof(struct NODO));
27
28     if( (*bicola)->primero == NULL ) {
29         temp->elemento.num = dato;
30         temp->p_anterior = NULL;
31         temp->p_siguiente = NULL;
32         (*bicola)->primero = temp;
33         (*bicola)->ultimo = temp;
34     }else{
35         temp->elemento.num = dato;
36         temp->p_anterior = NULL;
37         temp->p_siguiente = (*bicola)->primero;
```

```

38     (*bicola)->primero->p_anterior = temp; |
39     (*bicola)->primero = temp;
40     };
41     (*bicola)->nodos += 1;
42 };
43 void insertDerBicola( struct BICOLA **bicola, int dato ){
44     struct NODO *temp = (struct NODO *) malloc(sizeof(struct NODO));
45
46     if( (*bicola)->primero == NULL ){
47         temp->elemento.num = dato;
48         temp->p_anterior = NULL;
49         temp->p_siguiente = NULL;
50         (*bicola)->primero = temp;
51         (*bicola)->ultimo = temp;
52     }else{
53         temp->elemento.num = dato;
54         temp->p_anterior = (*bicola)->ultimo;
55         temp->p_siguiente = NULL;
56         (*bicola)->ultimo->p_siguiente = temp;
57         (*bicola)->ultimo = temp;
58     };
59     (*bicola)->nodos += 1;
60 };
61 void eliminaIzqBicola( struct BICOLA **bicola ){
62     struct NODO *aBorrar;
63     if( (*bicola)->primero == NULL ){
64         printf( "No puede eliminar Nodos de una Bicola vacía." );
65     }else if( (*bicola)->nodos == 1 ){
66         free(*bicola);
67         inicializarBicola( bicola );
68     }else if( (*bicola)->nodos > 1 ){
69         aBorrar = (*bicola)->primero;
70         (*bicola)->primero->p_siguiente->p_anterior = NULL;
71         (*bicola)->primero = (*bicola)->primero->p_siguiente;
72         free(aBorrar);
73         (*bicola)->nodos -= 1;

```

```

75 }
76 void eliminaDerBicola( struct BICOLA **bicola ){
77     struct NODO *aBorrar;
78     if( (*bicola)->primero == NULL ){
79         printf( "No puede eliminar Nodos de una Bicola vacía." );
80     }else if( (*bicola)->nodos == 1 ){
81         free(*bicola);
82         inicializarBicola( bicola );
83     }else if( (*bicola)->nodos > 1 ){
84         aBorrar = (*bicola)->ultimo;
85         (*bicola)->ultimo->p_anterior->p_siguiente = NULL;
86         (*bicola)->ultimo = (*bicola)->ultimo->p_anterior;
87         free(aBorrar);
88         (*bicola)->nodos -= 1;
89     };
90 };
91 int tieneNodosLaBicola( struct BICOLA **bicola ){
92     int resp = 0;
93     if( (*bicola)->nodos != 0 )
94         resp = 1;
95     return resp;
96 };
97
98 void borraLaBicola( struct BICOLA **bicola ){
99     struct NODO *actual, *siguiente;
100     actual = (*bicola)->primero;
101     while( actual != NULL ){
102         siguiente = actual->p_siguiente;
103         free(actual);
104         actual = siguiente;
105     };
106     *bicola = NULL;
107 };
108
109 int cuantosNodosTieneLaBicola( struct BICOLA **bicola ){
110     return (*bicola)->nodos;
111 };
112
113 void copiarLaBicola( struct BICOLA **bicolaA, struct BICOLA **bicolaB ){
114     struct NODO *temp = (*bicolaA)->primero;
115     if( (*bicolaB)->primero != NULL ){
116         borraLaBicola( bicolaB );
117         inicializarBicola( bicolaB );
118     };
119     if( temp == NULL )printf( "La Bicola A no contiene Nodos, no se puede copiar nada." );
120     else{
121         while( temp != NULL ){
122             insertDerBicola( bicolaB, temp->elemento.num );
123             temp = temp->p_siguiente;
124         };
125     };
126 };
127
128
129
130 int sonIgualesLasBicolas( struct BICOLA **bicolaA, struct BICOLA **bicolaB ){
131     struct NODO *bica = (*bicolaA)->primero;
132     struct NODO *bicB = (*bicolaB)->primero;
133     int salirBucle=0, resp=1;
134     if( bica == NULL || bicB == NULL ){
135         resp = 0;
136     }
137     if( bica == NULL )
138         printf( "Debe de insertar antes algún Nodo en la Bicola A\n" );
139     if( bicB == NULL )
140         printf( "Debe de insertar antes algún Nodo en la Bicola B\n" );
141     }else{
142         while( !salirBucle ){
143             if( bica->elemento.num != bicB->elemento.num ){
144                 salirBucle = 1;
145             }
146             bica = bica->p_siguiente;
147             bicB = bicB->p_siguiente;
148         };
149     };
150     return resp;
151 };

```



```

143     resp = 0;
144     salirBucle = 1;
145 }else{
146     bica = bica->p_siguiente;
147     bicB = bicB->p_siguiente;
148     if( (bica != NULL && bicB == NULL) || (bica == NULL && bicB != NULL) ){
149         resp = 0;
150         salirBucle = 1;
151     }else if( (bica == NULL && bicB == NULL) ){
152         salirBucle = 1;
153     };
154 };
155 };
156 };
157 return resp;
158 };
159
160 void imprimeBicola( struct BICOLA **bicola ){
161     struct NODO *bic = (*bicola)->primero;
162     if( bic == NULL )
163         printf( "La Bicola no contiene Nodos." );
164     else{
165         printf( "Su Bicola contiene: " );
166         while( bic != NULL ){
167             printf( "%i, ", bic->elemento.num );
168             bic = bic->p_siguiente;
169         };
170         printf( "\n\n" );
171     };
172 };
173
174 int main(){
175     setlocale(LC_CTYPE, "Spanish");
176     enum opciones( salir, insertIzq, insertDer, eliminaIzq, eliminaDer, impIzq, impDer, impTodos, quedanNodos, cuantosNodosHay, copiaBicola, sonBicolasIguales, borraBicola ) opc;
177     struct BICOLA *bicolaA;
178     struct BICOLA *bicolaB;
179     int eleccion, nuevoDato;

```

```

179     inicializarBicola( &bicolaA );
180     inicializarBicola( &bicolaB );
181     do{
182         printf( "\n\nIndique que desea hacer con los Nodos de la Bicola:\n\n" );
183         printf( " 1. Añadir un Nodo por la izquierda\n" );
184         printf( " 2. Añadir un Nodo por la derecha\n" );
185         printf( " 3. Eliminar el primer Nodo\n" );
186         printf( " 4. Eliminar el ultimo Nodo\n" );
187         printf( " 5. Mostrar el primer Nodo\n" );
188         printf( " 6. Mostrar el ultimo Nodo\n" );
189         printf( " 7. Muestra todos los Nodos\n" );
190         printf( " 8. %cQuedan Nodos?\n",168 );
191         printf( " 9. Cuantos Nodos hay?\n" );
192         printf( "10. Copiar BicolaA a una nueva BicolaB\n" );
193         printf( "11. %cBicola A es igual que BicolaB?\n",168 );
194         printf( "12. Borrar la Bicola A\n\n" );
195         printf( " 0. Salir del programa.\n" );
196         do{
197             scanf( "%i", &eleccion );
198         }while( eleccion < 0 && eleccion > 12 );
199         opc = (enum opciones)(eleccion);
200         printf( "\n\n" );
201         switch( opc ){
202             case insertIzq:
203                 printf( "Introduzca el número entero que contendrá el nuevo Nodo de la Bicola: " );
204                 scanf( "%i", &nuevoDato );
205                 insertIzqBicola( &bicolaA, nuevoDato );
206                 break;
207             case insertDer:
208                 printf( "Introduzca el número entero que contendrá el nuevo Nodo de la Bicola: " );
209                 scanf( "%i", &nuevoDato );
210                 insertDerBicola( &bicolaA, nuevoDato );
211                 break;
212             case eliminaIzq:
213                 eliminaIzqBicola( &bicolaA );
214                 break;

```



```

215     case eliminaDer:
216         eliminaDerBicola( &bicolaA );
217         break;
218     case impIzq:
219         if( tieneNodosLaBicola( &bicolaA ) )
220             printf( "El primer Nodo contiene un: %i\n", bicolaA->primero->elemento.num );
221         else
222             printf("La Bicola no contiene Nodos");
223         break;
224     case impDer:
225         if( tieneNodosLaBicola( &bicolaA ) )
226             printf( "El ultimo Nodo contiene un: %i\n", bicolaA->ultimo->elemento.num );
227         else
228             printf("La Bicola no contiene Nodos");
229         break;
230     case impTodos:
231         imprimeBicola( &bicolaA );
232         break;
233     case quedanNodos:
234         if( tieneNodosLaBicola( &bicolaA ) )
235             printf( "La Bicola contiene Nodos." );
236         else
237             printf( "La Bicola esta vacía." );
238         break;
239     case cuantosNodosHay:
240         printf( "La Bicola contiene %i Nodos.", cuantosNodosTieneLaBicola( &bicolaA ) );
241         break;
242     case copiaBicola:
243         copiarLaBicola( &bicolaA, &bicolaB );
244         break;
245     case sonBicolasIguales:
246         if( sonIgualesLasBicolas( &bicolaA, &bicolaB ) )
247             printf( "Las Bicolas son idénticas" );
248         else
249             printf( "Las Bicolas son diferentes" );
250         break;

```

```

251
252
253
254
255     case borrarBicola:
256         borraLaBicola( &bicolaA );
257         inicializarBicola( &bicolaA );
258         break;
259     case salir:
260         system("cls");
261         break;
262 };
}; while( opc != salir );
borraLaBicola( &bicolaA );
borraLaBicola( &bicolaB );
};

```

```
C:\Users\Loki6\OneDrive\Documentos\tercer semestre\Estructura de datos\segunda unidad\bicola.exe

Indique que desea hacer con los Nodos de la Bicola:

1. Añadir un Nodo por la izquierda
2. Añadir un Nodo por la derecha
3. Eliminar el primer Nodo
4. Eliminar el ultimo Nodo
5. Mostrar el primer Nodo
6. Mostrar el ultimo Nodo
7. Muestra todos los Nodos
8. ¿Quedan Nodos?
9. Cuantos Nodos hay?
10. Copiar BicolaA a una nueva BicolaB
11. ¿Bicola A es igual que BicolaB?
12. Borrar la Bicola A

0. Salir del programa.
1

Introduzca el número entero que contendrá el nuevo Nodo de la Bicola: 4

Indique que desea hacer con los Nodos de la Bicola:

1. Añadir un Nodo por la izquierda
2. Añadir un Nodo por la derecha
3. Eliminar el primer Nodo
4. Eliminar el ultimo Nodo
```

```
C:\Users\Loki6\OneDrive\Documentos\tercer semestre\Estructura de datos\segunda unidad\bicola.exe

8. ¿Quedan Nodos?
9. Cuantos Nodos hay?
10. Copiar BicolaA a una nueva BicolaB
11. ¿Bicola A es igual que BicolaB?
12. Borrar la Bicola A

0. Salir del programa.
2

Introduzca el número entero que contendrá el nuevo Nodo de la Bicola: 4

Indique que desea hacer con los Nodos de la Bicola:

1. Añadir un Nodo por la izquierda
2. Añadir un Nodo por la derecha
3. Eliminar el primer Nodo
4. Eliminar el ultimo Nodo
5. Mostrar el primer Nodo
6. Mostrar el ultimo Nodo
7. Muestra todos los Nodos
8. ¿Quedan Nodos?
9. Cuantos Nodos hay?
10. Copiar BicolaA a una nueva BicolaB
11. ¿Bicola A es igual que BicolaB?
12. Borrar la Bicola A

0. Salir del programa.
7
```

```
C:\Users\Loki6\OneDrive\Documentos\tercer semestre\Estructura de datos\segunda unidad\bicola.exe

0. Salir del programa.
7
Su Bicola contiene: 4, 4,

Indique que desea hacer con los Nodos de la Bicola:

1. Añadir un Nodo por la izquierda
2. Añadir un Nodo por la derecha
3. Eliminar el primer Nodo
4. Eliminar el ultimo Nodo
5. Mostrar el primer Nodo
6. Mostrar el ultimo Nodo
7. Muestra todos los Nodos
8. ¿Quedan Nodos?
9. Cuantos Nodos hay?
10. Copiar BicolaA a una nueva BicolaB
11. ¿Bicola A es igual que BicolaB?
12. Borrar la Bicola A

0. Salir del programa.
```

```
1  /*listas enlazadas*/
2  #include<iostream>
3  #include<conio.h>
4  #include<stdio.h>
5  #include<stdlib.h>
6  using namespace std;
7  struct Nodo
8  {
9      int dato;
10     Nodo *siguiente;
11 };
12 void menu();
13 void insertarLista(Nodo *&,int);
14 void mostrarLista(Nodo *listas);
15 void buscarLista(Nodo *,int);
16 void eliminarNodo(Nodo *&,int);
17 Nodo *lista=NULL;
18 int main()
19 {
20     menu();
21     getch();
22     return 0;
23 }
24 void menu()
25 {
26     int opcion;
27     do
28     {
29         cout<<"\t.:Menu:.\n";
30         cout<<"1. Insertar elemntos a la lista\n";
31         cout<<"2. Mostrar elemento de la lista\n";
32         cout<<"3. Buscar un elemento en lista\n";
33         cout<<"4. Eliminar un elemnto de la lista\n";
34         cout<<"5. Salir\n";
35         cout<<"\topcion: ";
36         cin>>opcion;
37         switch(opcion)
```

```

38 {
39     int dato;
40     case 1:
41         cout<<"digite un numero";
42         cin>>dato;
43         insertarLista(lista,dato);
44         cout<<"\n";
45         system("pause");
46         break;
47     case 2:
48         mostrarLista(lista);
49         cout<<"\n";
50         system("pause");
51     case 3:
52         cout<<"Digite el numero a buscar\n";
53         cin>>dato;
54         buscarLista(lista,dato);
55         cout<<"\n";
56         system("pause");
57         break;
58     case 4:
59         cout<<"digite el elemto a eliminar";
60         cin>>dato;
61         eliminarNodo(lista,dato);
62         cout<<"\n";
63         system("pause");
64     default:
65         cout<<"esa opcion no esta disponible";
66     }
67     system("cls");
68 }
69 while(opcion!=5);
70 }
71 void insertarLista(Nodo *&lista,int n)
72 {
73     Nodo *nuevo_nodo=new Nodo();

```

```

74     nuevo_nodo->dato=n;
75     Nodo *aux1=lista;
76     Nodo *aux2;
77     while((aux1!=NULL)&&(aux1->dato<n))
78     {
79         aux2=aux1;
80         aux1=aux1->siguiente;
81     }
82     if(lista==aux1)
83     {
84         lista=nuevo_nodo;
85     }
86     else
87     {
88         aux2->siguiente=nuevo_nodo;
89     }
90     nuevo_nodo->siguiente=aux1;
91     cout<<"\tElemento"<<n<<"insertado a lista correctamente\n";
92 }
93 void mostrarLista(Nodo *lista)
94 {
95     Nodo *actual=new Nodo();
96     actual=lista;
97     while(actual!=NULL)
98     {
99         cout<<actual->dato<<"->";
100         actual=actual->siguiente;
101     }
102 }
103 void buscarLista(Nodo *lista,int n)
104 {
105     bool band=false;
106     Nodo *actual=new Nodo();
107     actual=lista;
108     while((actual!=NULL)&&(actual->dato<=n) )
109     {

```

```

110     if(actual->dato==n)
111     {
112         band=true;
113     }
114     actual=actual->siguiente;
115 }
116 if(band==true)
117 {
118     cout<<"Elemento"<<n<<"SI a sido encontrado en lista\n";
119 }
120 else
121 {
122     cout<<"Elemento"<<"No a sido encontrado en lista\n";
123 }
124 }
125 void eliminarNodo(Nodo * &lista,int n)
126 {
127     if(lista!=NULL)
128     {
129         Nodo *aux_borrar;
130         Nodo *anterior=NULL;
131         aux_borrar=lista;
132         while((aux_borrar!=NULL)&&(aux_borrar->dato!=n))
133         {
134             anterior=aux_borrar;
135             aux_borrar=aux_borrar->siguiente;
136         }
137         if(aux_borrar==NULL)
138         {
139             cout<<"El elemnto no ha sido encontrado";
140         }
141         else if(anterior==NULL)
142         {
143             lista=lista->siguiente;
144             delete aux_borrar;
145         }
146         else
147         {
148             anterior->siguiente=aux_borrar->siguiente;
149             delete aux_borrar;
150         }
151     }
152 }

```

```
C:\Users\Loki6\OneDrive\Documentos\tercer semestre\Estructura de datos\segunda unidad\listas_enlazadas.exe

.:Menu:.
1. Insertar elemntos a la lista
2. Mostrar elemento de la lista
3. Buscar un elemento en lista
4. Eliminar un elemnto de la lista
5. Salir
    opcion: 1
digite un numero
4
    Elemento 4 insertado a lista correctamente
Presione una tecla para continuar . . .
```

```
C:\Users\Loki6\OneDrive\Documentos\tercer semestre\Estructura de datos\segunda unidad\listas_enlazadas.exe

.:Menu:.
1. Insertar elemntos a la lista
2. Mostrar elemento de la lista
3. Buscar un elemento en lista
4. Eliminar un elemnto de la lista
5. Salir
    opcion: 2
4->5->6->7->9->
Presione una tecla para continuar . . .
```

```
C:\Users\Loki6\OneDrive\Documentos\tercer semestre\Estructura de datos\segunda unidad\listas_enlazadas.exe

.:Menu:.
1. Insertar elemntos a la lista
2. Mostrar elemento de la lista
3. Buscar un elemento en lista
4. Eliminar un elemnto de la lista
5. Salir
    opcion: 2
2->4->6->8->9->
Presione una tecla para continuar . . .
Digite el numero a buscar
8
Elemento 8 SI a sido encontrado en lista
Presione una tecla para continuar . . .
```

```
C:\Users\Loki6\OneDrive\Documentos\tercer semestre\Estructura de datos\segunda unidad\listas_enlazadas.exe

.:Menu:.
1. Insertar elemntos a la lista
2. Mostrar elemento de la lista
3. Buscar un elemento en lista
4. Eliminar un elemnto de la lista
5. Salir
    opcion: 2
2->4->6->9->
Presione una tecla para continuar . . .
```



```

1  #include <iostream>
2  using namespace std;
3  #define ASCENDENTE 1
4  #define DESCENDENTE 0
5  class nodo {
6      public:
7          nodo(int v, nodo *sig = NULL, nodo *ant = NULL) :
8              valor(v), siguiente(sig), anterior(ant) {}
9          private:
10             int valor;
11             nodo *siguiente;
12             nodo *anterior;
13             friend class lista;
14 };
15 typedef nodo *pnodo;
16 class lista {
17     public:
18         lista() : plista(NULL) {}
19         ~lista();
20         void Insertar(int v);
21         void Borrar(int v);
22         bool ListaVacía() { return plista == NULL; }
23         void Mostrar(int);
24         void Siguiente();
25         void Anterior();
26         void Primero();
27         void Ultimo();
28         bool Actual() { return plista != NULL; }
29         int ValorActual() { return plista->valor; }
30     private:
31         pnodo plista;
32 };
33 lista::~lista() {
34     pnodo aux;
35     Primero();
36     while(plista) {
37         aux = plista;

```

```

38     ..... plista = plista->siguiente;
39     ..... delete aux;
40 }
41 }
42
43 void lista::Insertar(int v) {
44     pnode nuevo;
45     Primero();
46     if(ListaVacía() || plista->valor > v) {
47         nuevo = new nodo(v, plista);
48         if(!plista) plista = nuevo;
49         else plista->anterior = nuevo;
50     }
51     else {
52
53         while(plista->siguiente && plista->siguiente->valor <= v) siguiente();
54
55         nuevo = new nodo(v, plista->siguiente, plista);
56         plista->siguiente = nuevo;
57         if(nuevo->siguiente) nuevo->siguiente->anterior = nuevo;
58     }
59 }
60
61 void lista::Borrar(int v) {
62     pnode nodo;
63
64     nodo = plista;
65     while(nodo && nodo->valor < v) nodo = nodo->siguiente;
66     while(nodo && nodo->valor > v) nodo = nodo->anterior;
67
68     if(!nodo || nodo->valor != v) return;
69
70
71     if(nodo->anterior)
72         ..... nodo->anterior->siguiente = nodo->siguiente;
73     if(nodo->siguiente)

```

```

74     nodo->siguiente->anterior = nodo->anterior;
75     delete nodo;
76 }
77
78 void lista::Mostrar(int orden) {
79     pnode nodo;
80     if(orden == ASCENDENTE) {
81         Primero();
82         nodo = plista;
83         while(nodo) {
84             cout << nodo->valor << "-> ";
85             nodo = nodo->siguiente;
86         }
87     }
88     else {
89         Ultimo();
90         nodo = plista;
91         while(nodo) {
92             cout << nodo->valor << "-> ";
93             nodo = nodo->anterior;
94         }
95     }
96     cout << endl;
97 }
98
99 void lista::Siguiete() {
100     if(plista) plista = plista->siguiente;
101 }
102
103 void lista::Anterior() {
104     if(plista) plista = plista->anterior;
105 }
106
107 void lista::Primero() {
108     while(plista && plista->anterior) plista = plista->anterior;
109 }

```

```
110
111 void lista::Ultimo() {
112     while(plista && plista->siguiente) plista = plista->siguiente;
113 }
114
115 int main() {
116     lista Lista;
117
118     Lista.Insertar(20);
119     Lista.Insertar(10);
120     Lista.Insertar(40);
121     Lista.Insertar(30);
122
123     Lista.Mostrar(ASCENDENTE);
124     Lista.Mostrar(DSCENDENTE);
125
126     Lista.Primer();
127     cout << "Primer: " << Lista.ValorActual() << endl;
128
129     Lista.Ultimo();
130     cout << "Ultimo: " << Lista.ValorActual() << endl;
131
132     Lista.Borrar(10);
133     Lista.Borrar(15);
134     Lista.Borrar(45);
135     Lista.Borrar(40);
136
137     Lista.Mostrar(ASCENDENTE);
138     Lista.Mostrar(DSCENDENTE);
139
140     return 0;
141 }
142
```

```
C:\Users\Loki6\OneDrive\Documentos\tercer semestre\Estructura de datos\segunda unidad\listas_doblementeenlazadas.exe
10-> 20-> 30-> 40->
40-> 30-> 20-> 10->
PrimerO: 10
Ultimo: 40
20-> 30->
30-> 20->

-----
Process exited after 2.612 seconds with return value 0
Presione una tecla para continuar . . .
```

```

1  /*listas circulares*/
2  #include <iostream>
3  using namespace std;
4
5  class nodo {
6      public:
7          nodo(int v, nodo *sig = NULL) {
8              valor = v;
9              siguiente = sig;
10         }
11
12         private:
13             int valor;
14             nodo *siguiente;
15
16         friend class lista;
17     };
18
19     typedef nodo *pnodo;
20
21     class lista {
22     public:
23         lista() { actual = NULL; }
24         ~lista();
25
26         void Insertar(int v);
27         void Borrar(int v);
28         bool ListaVacía() { return actual == NULL; }
29         void Mostrar();
30         void Siguiente();
31         bool Actual() { return actual != NULL; }
32         int ValorActual() { return actual->valor; }
33
34     private:
35         pnodo actual;
36     };

```

```

38 lista::~~lista() {
39     pnode nodo;
40
41     // Mientras la lista tenga más de un nodo
42     while(actual->siguiente != actual) {
43         // Borrar el nodo siguiente al apuntado por lista
44         nodo = actual->siguiente;
45         actual->siguiente = nodo->siguiente;
46         delete nodo;
47     }
48     // Y borrar el último nodo
49     delete actual;
50     actual = NULL;
51 }
52
53 void lista::Insertar(int v) {
54     pnode Nodo;
55
56     // Creamos un nodo para el nuevo valor a insertar
57     Nodo = new nodo(v);
58
59     // Si la lista está vacía, la lista será el nuevo nodo
60     // Si no lo está, insertamos el nuevo nodo a continuación del apuntado
61     // por lista
62     if(actual == NULL) actual = Nodo;
63     else Nodo->siguiente = actual->siguiente;
64     // En cualquier caso, cerramos la lista circular
65     actual->siguiente = Nodo;
66 }
67
68 void lista::Borrar(int v) {
69     pnode nodo;
70
71     nodo = actual;

```

```

74 do {
75     if(actual->siguiente->valor != v) actual = actual->siguiente;
76 } while(actual->siguiente->valor != v && actual != nodo);
77 // Si existe un nodo con el valor v:
78 if(actual->siguiente->valor == v) {
79     // Y si la lista sólo tiene un nodo
80     if(actual == actual->siguiente) {
81         // Borrar toda la lista
82         delete actual;
83         actual = NULL;
84     }
85     else {
86         // Si la lista tiene más de un nodo, borrar el nodo de valor v
87         nodo = actual->siguiente;
88         actual->siguiente = nodo->siguiente;
89         delete nodo;
90     }
91 }
92 }
93
94 void lista::Mostrar() {
95     pnode nodo = actual;
96
97     do {
98         cout << nodo->valor << "-> ";
99         nodo = nodo->siguiente;
100     } while(nodo != actual);
101
102     cout << endl;
103 }
104
105 void lista::Siguiete() {
106     if(actual) actual = actual->siguiente;
107 }

```

```

109 int main() {
110     lista Lista;
111
112     Lista.Insertar(20);
113     Lista.Insertar(10);
114     Lista.Insertar(40);
115     Lista.Insertar(30);
116     Lista.Insertar(60);
117
118     Lista.Mostrar();
119
120     cout << "Lista de elementos:" << endl;
121     Lista.Borrar(10);
122     Lista.Borrar(30);
123
124     Lista.Mostrar();
125
126     cin.get();
127     return 0;
128 }

```



```
C:\Users\Loki6\OneDrive\Documentos\tercer semestre\Estructura de datos\segunda unidad\listas_circulares.exe
20-> 60-> 30-> 40-> 10->
Lista de elementos:
60-> 40-> 20->
```