```cpp
#include "TDANodoDoble.h"
#include <iostream>

using namespace std;
/*Nodo Incio de la Lista*/
struct TDAListaDoble{
    Nodo *inicio;
    Nodo *actual;
} ;

TDAListaDoble *Inicilizar(){
    TDAListaDoble *c;
    c = new TDAListaDoble;
    c->actual=NULL;
    c->inicio = NULL;
    return c;
}

/*  Crea la Lista,  moviendo siempre la posición actual al elemento ingresado*/
TDAListaDoble *crearLista(TDAListaDoble *c, int numero){
    Nodo *aux;
    aux =  crearNodo(numero) ;
    if (c->inicio==NULL){
        c->inicio = c->actual = aux;
    }
    else {
        aux->ant=c->actual;
        c->actual->sig = aux;
        c->actual=aux;
    }
    return c;
}

TDAListaDoble *addIzquierda(TDAListaDoble *c, int numero){
    Nodo *aux;
    aux =  crearNodo(numero);

    if (c->inicio==NULL){
        c->inicio = c->actual = aux;
    }
    else {
        aux->sig = c->inicio;
        c->inicio->ant=aux;
        c->inicio=aux;
    }
    return c;
}

void ImprimirLista(TDAListaDoble *c){
    if (c->inicio==NULL){
        cout<<"\n"<<"La lista no está creada";
    }
    else {
        Nodo *aux;
        aux=c->inicio;

        do{
            cout<<"\n"<<aux->info;
            aux=aux->sig;
        }while (aux != NULL);
    }
    cout<<"\n";
}

int cantElementEnLista(int numero,TDAListaDoble *c){
    int cont = 0;
```

```
67:     if (c->inicio==NULL){
68:         return 0;
69:     }
70:     else {
71:         Nodo *aux;
72:         aux=c->inicio;
73:
74:         do{
75:             if(aux->info == numero)
76:                 cont ++;
77:             aux=aux->sig;
78:         }while (aux != NULL);
79:     }
80:     return cont;
81: }
82:
83: void borrarElemento(TDAListaDoble *c, Nodo *nodBorrar){
84:     Nodo *ant;
85:     c->actual = c->inicio;
86:     while (c->actual != nodBorrar){
87:         ant = c->actual;
88:         c->actual=c->actual->sig;
89:     }
90:
91:     if (c->inicio==nodBorrar){
92:         c->actual=c->inicio=c->inicio->sig;
93:     }
94:     else {
95:         ant->sig=nodBorrar->sig;
96:     }
97:     delete(nodBorrar);
98: }
99:
100:
101: void borrarIndex(TDAListaDoble *c, int pos){
102:     int cont = 1;
103:     if (c->inicio==NULL){
104:         cout<<"\n"<<"La lista no está creada";
105:     }
106:     else {
107:         Nodo *aux;
108:         aux=c->inicio;
109:
110:         do{
111:             if(cont == pos){
112:                 borrarElemento(c, aux);
113:                 return;
114:                 cout << "Supuestamente borrado\n";
115:             }
116:             aux=aux->sig;
117:             cont++;
118:         }while (aux != NULL);
119:     }
120: }
121:
122: int longitud(TDAListaDoble *c){
123:     int cont = 0;
124:     if (c->inicio==NULL){
125:         return 0;
126:     }
127:     else {
128:         Nodo *aux;
129:         aux=c->inicio;
130:
131:         do{
132:             cont ++;
```

```
133:             aux=aux->sig;
134:         }while (aux != NULL);
135:     }
136:     return cont;
137: }
138:
139: int esVacia(TDAListaDoble *c){
140:     if(c->inicio == c->actual)
141:         return 1;
142:     else
143:         return 0;
144: }
145:
146: TDAListaDoble *insertar(TDAListaDoble *c, int pos, int info){
147:     // Inserta un elemento info en la ListaDoble c, en la posicion pos
148:     // NOTA: Si se ingresa una posicion mayor al tamaño de la lista c, este elemento se inserta
149:     int cont = 0;
150:     if(pos == 1){
151:         c = addIzquierda(c, info);
152:     }
153:     else if (c->inicio==NULL || pos>longitud(c)){
154:         c = crearLista(c,info);
155:     }
156:     else{
157:         Nodo *aux, *aux2 = crearNodo(info);
158:         aux=c->inicio;
159:         do{
160:             cont++;
161:             cout<<"\n"<<aux->info;
162:             if(cont == pos-1){
163:                 aux2->sig = aux->sig;
164:                 aux->sig = aux2;
165:                 aux2->ant = aux;
166:                 aux2->sig->ant = aux2;
167:             }
168:             aux=aux->sig;
169:         }while (aux != NULL);
170:     }
171:     return c;
172: }
173:
174: TDAListaDoble *ordenar(TDAListaDoble *c){   // ordenamiento por inserción
175:     struct TDAListaDoble *c2 = Inicilizar();
176:     int tamanio = longitud(c), sw = 0, cont;
177:     c2 = crearLista(c2, c->inicio->info);
178:     borrarElemento(c, c->inicio);
179:     ImprimirLista(c2);
180:     Nodo *aux;
181:     for(int i=0; i<tamanio; i++){
182:         while(c->inicio != NULL){
183:             cont = 1;
184:             aux = c2->inicio;
185:             sw = 0;
186:             do{
187:                 cout << aux->info << " > " << c->inicio->info << endl;
188:                 if(aux->info > c->inicio->info){
189:                     c2 = insertar(c2,cont,c->inicio->info);
190:                     borrarElemento(c,c->inicio);
191:                     sw = 1;
192:                 }
193:                 aux=aux->sig;
194:                 cont++;
195:             }while (aux != NULL && sw != 1);
196:             if(sw == 0){
197:                 c2 = crearLista(c2, c->inicio->info);
198:                 borrarElemento(c,c->inicio);
```

```
            }
        }
    }
    system("pause");
    return c2;
}

int buscar(int elemento, TDAListaDoble *c){
    int encontrado;
    if (c->inicio==NULL){
        encontrado= 0;
    }
    else{
        Nodo *aux=c->inicio;
        while(aux!=NULL && !encontrado){
            if(aux->info==elemento){
                encontrado=1;
            }
            aux = aux ->sig;
        }
    }
    return encontrado;
}

void limpiarLista(TDAListaDoble *c){
    int lon = longitud(c);
    for(int i=0; i<lon ;i++){
        borrarElemento(c,c->inicio);
    }
}
```