Práctica 4

Listas Simple y Ordenada

1. Ejecute el programa ***Dev-C++***.
2. En el menú ***Archivo*** seleccione la opción ***Nuevo*** y a continuación ***Archivo Fuente***.
3. En el nuevo documento transcriba el código del siguiente programa, en el cual se definen las clases nodo y lista con sus atributos y métodos respectivos y se cuenta con un menú para realizar operaciones de inserción y eliminación en una lista simple de números enteros:

```
#include <iostream>
#include <iomanip>
using namespace std;

template<class T>
class Nodo {
    public: T info;
            Nodo<T> *sig;
            Nodo(const T);
};

template<class T>
class Lista {
   private: Nodo <T> *le;

    public: Lista ();
            ~ Lista ();
            void Ins_Ini(const T);
            void Ins_Fin(const T);
            T Eli_Ini();
            T Eli_Fin();
            void Ver();
            int  Vacia();
};

template<class T>
Nodo<T>::Nodo(const T v) {
   info = v;
   sig = NULL;
}

template<class T>
Lista<T>::Lista() {
```

```
    le = NULL;
}

template<class T>
Lista<T>::~Lista() {
   Nodo <T>*sale;
   T val;
   while(le) {
        sale=le;
        le=le->sig;
        val=sale->info;
        delete(sale);
        cout<<"Bloque de memoria liberado: "<<val<<endl;
    }
    delete le;
}

template<class T>
void Lista<T>::Ins_Ini(const T v) {
    Nodo <T>*nvo = new Nodo<T>(v);
    nvo->sig=le;
    le=nvo;
}

template<class T>
void Lista<T>::Ins_Fin(const T v) {
    Nodo <T>*nvo = new Nodo<T>(v);
    Nodo <T>*tmp;
    if(Vacia()) le=nvo;
    else {
        tmp=le;
        while(tmp->sig!=NULL) tmp=tmp->sig;
        tmp->sig=nvo;
    }
}

template<class T>
T Lista<T>::Eli_Ini() {
    Nodo <T>*sale;
    T val;
    if (Vacia()) val= 0;
    else {
      sale=le;
      le=sale->sig;
      val=sale->info;
      delete sale;
```

```
    }
    return val;
}

template<class T>
T Lista<T>::Eli_Fin() {
    Nodo <T>*sale, *aux;
    T val;
    if(Vacia()) return 0;
    else {
        sale=le;
        if(le->sig==NULL) le=NULL;
        else {
            while(sale->sig!=NULL) {
                aux=sale;
                sale=sale->sig;
            }
            aux->sig=NULL;
        }
        val=sale->info;
        delete sale;
        return val;
    }
}

template<class T>
int Lista<T>::Vacia()  {
    return le==NULL ? 1 : 0;
}

template<class T>
void Lista<T>::Ver()  {
   Nodo <T>*tmp = le;
    if(Vacia()) cout<<"Lista vacia.";
    else {
        cout <<endl<< "La lista es: ";
        while (tmp) {
            cout << setw(5) << tmp->info;
            tmp = tmp->sig;
        }
    }
    cout <<endl;
}

int main(void) {
    Lista <int> lista;
```

```
    int valor;
    char opc;
    cout<<endl<<"Implementacion de una Lista Simple"<<endl;
    do {
       cout<<"\n1.- Insertar Inicio\n2.- Insertar Final";
       cout<<"\n3.- Eliminar Inicio\n4.- Eliminar Final";
       cout<<"\n5.- Ver lista actual\n6.- Salir\n\nOpcion: ";
       cin>>opc;
       switch(opc) {
          case '1': cout<<endl<<"Introduce un valor diferente de cero: ";
                    cin>>valor;
                    lista.Ins_Ini(valor);
                    break;
          case '2': cout<<endl<<"Introduce un valor diferente de cero: ";
                    cin>>valor;
                    lista.Ins_Fin(valor);
                    break;
          case '3': valor= lista.Eli_Ini();
                    if(valor ==0) cout<<endl<<"Lista vacia."<<endl;
                    else cout<<endl<<"Se elimino dato: "<< valor <<endl;
                    break;
          case '4': valor = lista.Eli_Fin();
                    if(valor ==0) cout<<endl<<"Lista vacia."<<endl;
                    else cout<<endl<<"Se elimino dato: "<< valor <<endl;
                    break;
          case '5': lista.Ver();
                    break;
       }
    } while(opc!='6');
}
```

4. Ahora localice el icono de ***Compilar*** que está al inicio de la barra de herramientas y oprímalo, proporcione el nombre de ***ejercicio 4A*** y elija el destino que guste para guardarlo.
5. Ahora ejecútelo y agregue diferentes valores enteros por la izquierda o la derecha según se elija la opción de inserción, ahora elimine algunos de valores de la lista por ambos extremos, observe el comportamiento de la lista.
6. En el menú ***Archivo*** seleccione la opción ***Nuevo*** y a continuación ***Archivo Fuente***.
7. En el nuevo documento transcriba el código del programa anterior, en el cual se define una lista doble, los cambios son mínimos y están resaltados para su fácil identificación:

```
#include<iostream>
#include <iomanip>
using namespace std;

template <class T>
class Nodo {
```

```
   public: T info;
           Nodo <T> *ant,*sig;
           Nodo(const T);
};

template<class T>
class Lista_Doble {
   private: Nodo <T> *le;
   public:  Lista_Doble ();
           ~ Lista_Doble ();
           void Ins_Ini(const T);
           void Ins_Fin(const T);
           T Eli_Ini();
           T Eli_Fin();
           void Ver();
           int  Vacia();
};

template <class T>
Nodo<T>::Nodo(const T v) {
    info = v;
    ant  = NULL;
    sig  = NULL;
}

template<class T>
Lista_Doble<T>::Lista_Doble() {
   le = NULL;
}

template <class T>
Lista_Doble<T>::~Lista_Doble () {
    Nodo <T>*sale;
    T val;
    while(le) {
        sale=le;
        le=le->sig;
        val=sale->info;
        delete(sale);
        cout<<"Bloque de Memo Liberado: "<<val<<endl;
    }
    delete le;
}

template <class T>
void Lista_Doble<T>::Ins_Ini(const T v) {
```

```
    Nodo <T>*nvo = new Nodo<T>(v);
    if(Vacia()) le=nvo;
    else {
        nvo->sig=le;
        le->ant=nvo;
        le=nvo;
    }
}

template <class T>
void Lista_Doble<T>::Ins_Fin(const T v) {
    Nodo <T>*nvo = new Nodo<T>(v);
    Nodo <T>*tmp;
    if(Vacia()) le=nvo;
    else {
        tmp=le;
        while(tmp->sig!=NULL) tmp=tmp->sig;
            tmp->sig=nvo;
            nvo->ant=tmp;
    }
}

template <class T>
T Lista_Doble<T>::Eli_Ini() {
    Nodo <T>*sale;
    T val;
    if(Vacia()) return 0;
    else {
        sale=le;
        if(sale->sig==NULL) le=NULL;
        else {
            le=sale->sig;
            le->ant=NULL;
            sale->sig=NULL;
        }
        val=sale->info;
        delete sale;
        return val;
    }
}

template <class T>
T Lista_Doble<T>::Eli_Fin() {
    Nodo <T>*sale,*aux;
    T val;
    if(Vacia()) return 0;
```

```
    else {
        sale=le;
        if(sale->sig==NULL) le=NULL;
        else {
            while(sale->sig!=NULL) {
                aux=sale;
                sale=sale->sig;
            }
            aux->sig=NULL;
            sale->ant=NULL;
        }
        val=sale->info;
        delete sale;
        return val;
    }
}

template <class T>
int Lista_Doble<T>::Vacia() {
    return le==NULL ? 1 : 0;
}

template <class T>
void Lista_Doble<T>::Ver() {
    Nodo <T>*tmp = le;
    if(Vacia()) cout<<endl<<"Lista vacia.";
    else {
        cout <<endl<<"La lista es: ";
        while (tmp) {
            cout << setw(5) << tmp->info;
            tmp = tmp->sig;
        }
    }
    cout <<endl;
}

int main(void) {
    Lista_Doble <int> le;
    int valor;
    char opc;
    cout<<endl<<" Implementacion de una Lista Doble"<<endl<<endl;
    do {
        cout<<"\n1.- Insertar Inicio\n2.- Insertar Final";
        cout<<"\n3.- Eliminar Inicio\n4.- Eliminar Final";
        cout<<"\n5.- Ver lista actual\n6.- Salir\n\nOpcion: ";
        cin>>opc;
```

```
        switch(opc) {
           case '1': cout<<endl<<"Introduce un valor diferente de cero: ";
                     cin>>valor;
                     le.Ins_Ini(valor);
                     break;
           case '2': cout<<endl<<"Introduce un valor diferente de cero: ";
                     cin>>valor;
                     le.Ins_Fin(valor);
                     break;
           case '3': valor =le.Eli_Ini();
                     if(valor ==0) cout<<endl<<"Lista vacia."<<endl;
                     else cout<<endl<<"Se elimino dato: "<< valor <<endl;
                     break;
           case '4': valor =le.Eli_Fin();
                     if(valor ==0) cout<<endl<<"Lista vacia."<<endl;
                     else cout<<endl<<"Se elimino dato: "<< valor <<endl;
                     break;
           case '5': le.Ver();
                     break;
        }
     } while(opc!='6');
}
```

8. Ahora localice el icono de ***Compilar*** que está al inicio de la barra de herramientas y oprímalo, proporcione el nombre de ***ejercicio 4B*** y elija el destino que guste para guardarlo.
9. Ahora ejecútelo y observe que el comportamiento es idéntico al del programa anterior.
10. En el menú ***Archivo*** seleccione la opción ***Nuevo*** y a continuación ***Archivo Fuente***.
11. En el nuevo documento transcriba el código del primer programa, y ahora vamos a modificarlo para manipular una lista ordenada, los cambios corresponden a los métodos de insertar y eliminar, y están resaltados para su fácil identificación:

```
#include <iostream>
#include <iomanip>
using namespace std;

template<class T>
class Nodo {
   public: T info;
           Nodo<T> *sig;
           Nodo(const T);
};

template<class T>
class Lista {
   private: Nodo <T> *le;
   public:  Lista();
```

```
            ~Lista();
            void Insertar(const T);
            T Eliminar(const T);
            void Ver();
            int Vacia();
};

template<class T>
Nodo<T>::Nodo(const T v) {
   info = v;
   sig = NULL;
}

template<class T>
Lista<T>::Lista() {
   le = NULL;
}

template<class T>
Lista<T>::~Lista() {
   Nodo <T>*sale;
   T val;
   while(le) {
      sale=le;
      le=le->sig;
      val=sale->info;
      delete(sale);
      cout<<"Bloque de memoria liberado: "<<val<<endl;
   }
   delete le;
}

template<class T>
void Lista<T>::Insertar(const T v) {
   Nodo <T>*tmp;
   if (Vacia() || le->valor > v) {
      Nodo <T>*nvo = new Nodo<T>(v);
      nvo->sig = le;
      le = nvo;
   }
   else {
      tmp = le;
      while(tmp->sig && tmp->sig->valor <= v)  tmp = tmp->sig;
      Nodo <T>*nvo = new Nodo<T>(v);
      nvo->sig = tmp->sig;
      tmp->sig = nvo;
```

```
    }
    cout<<"Hemos insertado el valor. "<<v<<endl;
    Ver();
}

template<class T>
T Lista<T>::Eliminar(const T v) {
    Nodo <T>*tmp, *nodo;
    nodo = le;
    tmp = NULL;
    while(nodo && nodo->info < v) {
        tmp = nodo;
        nodo = nodo->sig;
    }
    if(!nodo || nodo->info != v) return 0;
    else {
        if(nodo==le)
            le = nodo->sig;
        else
            tmp->sig = nodo->sig;
        delete nodo;
    }
    return v;
}

template<class T>
int Lista<T>::Vacia()  {
      return le==NULL ? 1 : 0;
}

template<class T>
void Lista<T>::Ver() {
    Nodo<T> *tmp=le;
    if(Vacia()) cout << "La lista esta Vacia.";
    else {
        cout << "La lista actual es: ";
        while(tmp) {
            cout << setw(5) << tmp->info;
            tmp = tmp->sig;
        }
    }
    cout << endl;
}

int main() {
    Lista<int> le;
```

```
    int valor, band;
    char opc;
    cout<<endl<<"Implementacion de una Lista Ordenada"<<endl;
    do {
       cout<<"\n1.- Insertar \n2.- Eliminar \n3.- Ver lista actual\n4.- Salir\n\n";
       cout<<"Opcion: ";
       cin>>opc;
       switch(opc) {
          case '1': cout<<endl<<"Introduce un valor: ";
                    cin>>valor;
                    le.Insertar(valor);
                    break;
          case '2': cout<<endl<<"Introduce un valor a eliminar: ";
                    cin>>valor;
                    band=le.Eliminar(valor);
                    if(band==0) cout<<endl<<"El dato no se encuentra." <<endl;
                    else cout<<endl<<"El dato ha sido eliminado." <<endl;
                    le.Ver();
                    break;
          case '3': le.Ver();
                    break;
         }
     } while(opc!= '4');
}
```

12. Ahora localice el icono de ***Compilar*** que está al inicio de la barra de herramientas y oprímalo, proporcione el nombre de ***ejercicio 4C*** y elija el destino que guste para guardarlo.
13. Ahora ejecútelo y agregue diferentes valores enteros, observe como crece la lista de manera ordenada, observe que acepta valores duplicados.
14. Ahora elimine algunos de valores de la lista, intente eliminar valores que no estén en la lista.
15. Se deja como desafío al alumno que modifique el método ***Insertar*** para que impida el registro de valores duplicados.
16. Fin de la Práctica.