Programación 3

Facultad de Ingeniería y Ciencias Naturales

Instructor: Ernesto Enrique García Ramos

Contacto: egarcia97.r@gmail.com

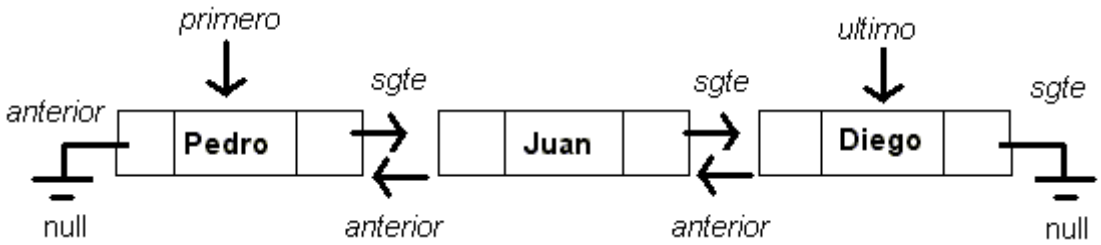
gr15i04001@usonsonate.edu.sv

Guía 3: Listas doblemente enlazadas

Objetivos:

- Facilitar códigos desarrollados sobre listas doblemente enlazadas
- Implementar listas dobles enlazadas en aplicaciones prácticas

Una lista doblemente enlazada es una lista lineal en la que cada nodo tiene dos enlaces, uno al nodo siguiente, y otro al anterior. Pueden recorrerse en ambos sentidos a partir de cualquier nodo, esto es porque a partir de cualquier nodo, siempre es posible alcanzar cualquier nodo de la lista, hasta que se llega a uno de los extremos.



Debido a su manejo de direcciones las listas doblemente enlazadas son más rápidas al momento de desplazarnos a una posición determinada de un elemento (Nodo). Ejemplo de ellos sería que nos encontremos en el último elemento de la lista (Diego) y necesitemos obtener el elemento anterior (Juan), la forma convencional de listas simples seria:

1. Posicionarnos al inicio de la lista
2. Hacer recorridos hacia adelante hasta encontrar el elemento deseado

Por otro lado, con listas doblemente enlazadas, que estén ordenadas, se puede simplificar a apuntar al elemento anterior. Evitando el uso de direccionamientos de memoria, agilizando la búsqueda.

Clase Nodo

```
class Nodo{
    private:
        int Valor;
        Nodo *Siguiente;
        Nodo *Anterior;
    public:
        Nodo(int v, Nodo *sig=NULL, Nodo *ant=NULL){
            this->Valor = v;
            this->Siguiente = sig;
            this->Anterior = ant;
        }
```

```
        friend class ListaDoble;
};typedef Nodo *p;
```

La clase Lista

En la lista doble los nodos se organizan de modo que cada uno apunta al siguiente, hasta llegar a NULL, y el nodo anterior apunta al nodo anterior, hasta llegar a NULL.

El recorrido se puede hacer desde el inicio al final de la lista o desde el final hacia el inicio de ella.

Principales diferencias a nivel de código con las listas simples:

La Primera diferencia es que ahora se manejan los direccionamientos mediante un único puntero (Actual) ya no usando el puntero Primero.

Es necesario que los métodos y funciones básicos cambien para adaptarse a usar solamente un puntero nodo que es el Actual.

Lista Simple

```
27  bool ListaVacia(){
28      return (this->primero==NULL);
29  }
30
31  void Primero(){
32      this->actual = this->primero;
33  }
34
35  void Siguiente(){
36      if(this->actual->Siguiente!=NULL){
37          this->actual = this->actual->Siguiente;
38      }
39  }
40
```

Lista Doble

```
27  bool ListaVacia(){
28      return (this->actual==NULL);
29  }
30
31  void Primero(){
32          while(this->actual && this->actual->Anterior){
33              this->Anterior();
34          }
35  }
36
37  void Siguiente(){
38      if(this->actual!=NULL){
39          this->actual = this->actual->Siguiente;
40      }
41  }
```

El método insertar (al final de la lista) tiene un leve cambio que es al momento de insertar un dato en una lista no vacía, en este caso el nuevo nodo a insertar en su posición anterior apunta al nodo de la lista actual, haciendo así un desplazamiento hacia atrás del nodo actual.

```
57      void Insertar(int valor){
58          pNodo nuevo = new Nodo(valor);
59          if(this->ListaVacia())
60          {
61              this->actual = nuevo;
62          }
63          else
64          {
65              this->Final();
66              this->actual->Siguiente = nuevo;
67              nuevo->Anterior = this->actual;
68          }
69      }
```

Si se desea insertar en una posición determinada ya sea en medio es de tener en cuenta las direcciones que se añadirían al nuevo nodo.

Código

```
class ListaDoble{
    private:
        p elemento;
        bool ListaVacia(){
            return (this->elemento==NULL);
        }
    public:
        ///Constructor
        ListaDoble(){
            this->elemento = NULL;
        }
        ~ListaDoble(){
            this->Primero();
            while (this->elemento){
                p aux = this->elemento;
                this->Siguiente();
                delete aux;
            }
        }
///Metodos de navegacion
        void Anterior(){
            if (this->elemento)
                this->elemento = this->elemento->Anterior;
        }
        void Siguiente(){
            if (this->elemento)
                this->elemento = this->elemento->Siguiente;
        }
        void Primero(){
            if (!this->ListaVacia()){
                while (this->elemento->Anterior){
                    this->Anterior();
                }
            }
        }
        void Ultimo(){
            if (!this->ListaVacia()){
                while (this->elemento->Siguiente){
                    this->Siguiente();
                }
            }
        }
void Insertar(int v){
            this->Ultimo();
            p nuevo = new Nodo(v);
            if (this->ListaVacia()){
                this->elemento = nuevo;
            }
            else{
                this->elemento->Siguiente = nuevo;
                nuevo->Anterior = this->elemento;
                this->elemento = nuevo;
            }
        }
```

```
        p Buscar(int v){
            bool encontrado=false;
            this->Primero();
            p aux = this->elemento;
            p retorno = NULL;
            while (aux && !encontrado){
                if (aux->Valor==v){
                    encontrado=true;
                    retorno = aux;
                }
                else{
                    aux = aux->Siguiente;
                }
            }
            return retorno;
        }
        void Eliminar(int v){
            p aux = this->Buscar(v);
            if (aux){
                ///Determinando si esta al principio
                if (aux->Anterior==NULL){
                    ///Al principio
                    this->elemento = this->elemento->Siguiente;
                    aux->Siguiente->Anterior = NULL;
                    delete aux;
                }
                else{
                    if (aux->Siguiente == NULL){
                        ///Ultimo
                        this->elemento = this->elemento->Anterior;
                        aux->Anterior->Siguiente = NULL;
                        delete aux;

                    }else{
                        if (aux->Anterior==NULL && aux-
>Siguiente==NULL){
                            ///Es unico
                            this->elemento = NULL;
                            delete aux;
                        }
                        else{
                            ///Esta en medio
                            this->elemento = aux->Siguiente;
                            aux->Siguiente->Anterior = aux-
>Anterior;
                            aux->Anterior->Siguiente = aux-
>Siguiente;
                            delete aux;
                        }
                    }
                }
            }
            else{
                cout << "No se pudo eliminar\n";
            }
        }
        void Modificar(int v, int n){
            p aux = this->Buscar(v);
            if (aux){
```

```
                aux->Valor = n;
            }
            else{
                cout << "No se pudo modificar porque no existe\n";
            }
        }
        void Mostrar(int orden){
            p aux;
            if (orden==1){
                ///1. Ascendente
                this->Primero();
                aux = this->elemento;
                cout << "NULL<-->";
                while (aux){
                    cout << aux->Valor;
                    cout << "<-->";
                    aux = aux->Siguiente;
                }
                cout << "NULL\n";
            }
            else{
                ///2. Descendente
                this->Ultimo();
                aux = this->elemento;
                cout << "NULL<-->";
                while (aux){
                    cout << aux->Valor;
                    cout << "<-->";
                    aux = aux->Anterior;
                }
                cout << "NULL\n";
            }
        }

};
```

Ejercicio

Implementar en el main los métodos