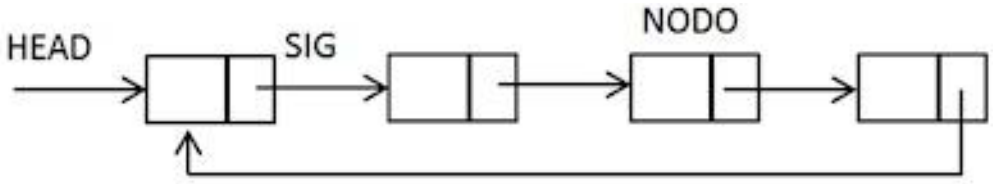
Asignatura: Algoritmia y Estructura de Datos
Profesor: D. Sc. Gerardo García Gil
2021-B
Alumno: José Rafael Ruiz Gudiño
Ingeniería en Desarrollo de Software
Centro de Enseñanza Técnica Industrial (CETI)

-Presentación

En términos muy generales, el área de estudio de estructuras de datos como disciplina de las ciencias computacionales (CS) es el almacenamiento y la manipulación de la información. Las estructuras de datos, siendo una disciplina dentro de la categoría de la computación teórica, estudian de manera más abstracta la relación entre una colección de datos y las operaciones que pueden ser aplicadas a dicha colección. A continuación, se hablará acerca de una de las estructuras de datos en el manejo de colecciones: las listas circulares.

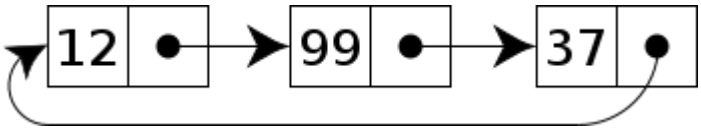
-Introducción

Una lista circularmente enlazada tiene el mismo tipo de nodos que una lista simple enlazada. Esto es, cada nodo en una lista circularmente enlazada tiene un apuntador siguiente y una referencia a un elemento. Pero no hay una cabeza o cola en la lista circularmente enlazada. En vez de tener que el apuntador del ultimo nodo sea null, en una lista circularmente enlazada, este apunta de regreso al primer nodo. Por lo tanto, no hay primer nodo o ultimo. Si se recorren los nodos de una lista circularmente enlazada desde cualquier nodo usando los apuntadores sig, se circulará a través de los nodos.



-Desarrollo

En una lista enlazada circular, el primer y el último nodo están unidos juntos. Esto se puede hacer tanto para listas enlazadas simples como para las doblemente enlazadas. Para recorrer una lista enlazada circular podemos empezar por cualquier nodo y seguir la lista en cualquier dirección hasta que se regrese hasta el nodo original. Desde otro punto de vista, las listas enlazadas circulares pueden ser vistas como listas sin comienzo ni fin. Este tipo de listas es el más usado para dirigir buffers para “ingerir” datos, y para visitar todos los nodos de una lista a partir de uno dado.



- Listas enlazadas simples circulares

Cada nodo tiene un enlace, similar al de las listas enlazadas simples, excepto que el siguiente nodo del último apunta al primero. Como en una lista enlazada simple, los nuevos nodos pueden ser solo eficientemente insertados después de uno que ya tengamos referenciado. Por esta razón, es usual quedarse con una referencia solamente al último elemento en una lista enlazada circular simple, esto nos permite rápidas inserciones al principio, y también permite accesos al primer nodo desde el puntero del último nodo.1

- Listas enlazadas doblemente circulares

En una lista enlazada doblemente circular, cada nodo tiene dos enlaces, similares a los de la lista doblemente enlazada, excepto que el enlace anterior del primer nodo apunta al último y el enlace siguiente del último nodo, apunta al primero. Como en una lista doblemente enlazada, las inserciones y eliminaciones pueden ser hechas desde cualquier punto con acceso a algún nodo cercano. Aunque estructuralmente una lista circular doblemente enlazada no tiene ni principio ni fin, un puntero de acceso externo puede establecer el nodo apuntado que está en la cabeza o al nodo cola, y así mantener el orden tan bien como en una lista doblemente enlazada.

-Implementación

El nodo típico es el mismo que para construir listas abiertas:

```
struct nodo {

  int dato;

  struct nodo *siguiente;

};
```

Lista es el tipo para declarar listas tanto abiertas como circulares. En el caso de las circulares, apuntará a un nodo cualquiera de la lista.

A pesar de que las listas circulares simplifiquen las operaciones sobre ellas, también introducen algunas complicaciones. Por ejemplo, en un proceso de búsqueda, no es tan sencillo dar por terminada la búsqueda cuando el elemento buscado no existe.

Por ese motivo se suele resaltar un nodo en particular, que no tiene por qué ser siempre el mismo. Cualquier nodo puede cumplir ese propósito, y puede variar durante la ejecución del programa.

Otra alternativa que se usa a menudo, y que simplifica en cierto modo el uso de listas circulares es crear un nodo especial de hará la función de nodo cabecera. De este modo, la lista nunca estará vacía, y se eliminan casi todos los casos especiales.

Código de lista circular con clases en C++

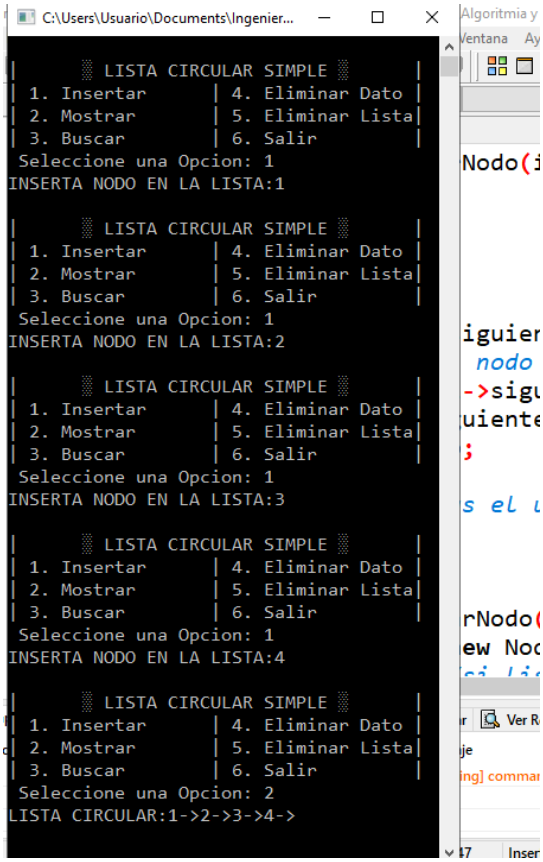
```
#include<iostream>
using namespace std;

class Nodo{
	private:
		int dato;
		Nodo *siguiente;
		friend class Lista;
	public:
		Nodo(int v, Nodo *sig = NULL){
			dato = v;
			siguiente = sig;
		}
};
typedef Nodo *pNodo;
class Lista{
	private:
		pNodo actual;
		pNodo ultimo;
	public:
		Lista(){
			actual = NULL;
		}
		~Lista();
		void insertarNodo(int n);
      void mostrarLista();
      void buscarNodo(int n);
      void borrarNodo(int n);
};
Lista::~Lista(){
	pNodo nodo;
	while(actual->siguiente!=actual) {  //mientras la lista tenga mas de un nodo
		//borrar el nodo siguiente al apuntado por la lista
		nodo=actual->siguiente;
		actual->siguiente=nodo->siguiente;
		delete nodo;
	}
	 //borrar ademas el ultimo nodo
	delete actual;
	actual=NULL;
}
void Lista::insertarNodo(int n){
	pNodo nuevo = new Nodo(n);
	if(!actual){ //si lista es null o esta vacio
		actual = nuevo; // la lista es igual al nuevo dato
		actual->siguiente = nuevo; //y lista->siguiente apunta a si mismo
		ultimo = nuevo; // y tambien el ultimo es igual a nuevo
	}
	else{ // si la lista circular no esta vacia
		ultimo->siguiente = nuevo; // ultimo->siguiente es igual al nuevo nodo
		nuevo->siguiente = actual; // y nuevo->siguiente es igual a lista para cerrar el circulo
		ultimo = nuevo; //ultimo es el nuevo nodo ingresado
	}
```

```
}
void Lista::mostrarLista(){
	pNodo nodo;
	nodo = actual;
	if(actual){ // si la lista no es null
	do{
		cout<<nodo->dato<<"->"; // se imprime el dato
		nodo = nodo->siguiente; // se mueve actual al siguiente nodo
	}while(actual!=nodo); //para que no entre en un bucle infiinito ya que todo esta enlazado
	cout<<endl;
  }else cout<<"\nLa lista esta vacia"<<endl;
}
void Lista::buscarNodo(int n){
	pNodo nodo = actual;
	bool encontrado = false;
	if(actual){
		do{
			if(nodo->dato==n){ // si actual->dato es igual al dato buscado
				cout<<"\nDato "<<nodo->dato<<" encontrado"<<endl;
				encontrado = true; // encontrado es igual a true
			}
			nodo = nodo->siguiente;   // actual se sigue desplazando
		}while(actual!=nodo&&encontrado!=true);

	if(!encontrado) cout<<"\nDato "<<n<<" no encontrado"<<endl;
	}else cout<<"\nLa lista esta vacia"<<endl;
}
void Lista::borrarNodo(int n){
	pNodo nodo= actual;
	if(actual){
	 do{
		//hace si no hay un valor y no hay nodo
		if(actual->siguiente->dato!=n)

		 actual=actual->siguiente;

	   }while(actual->siguiente->dato!=n && actual != nodo);

	 if(actual->siguiente->dato==n){
		if(actual == actual->siguiente){ // cuando es solo un nodo en la lista circular
			//borra toda la lista
			delete actual;
			actual = NULL;
		}
		else{
			nodo=actual->siguiente;
			actual->siguiente=nodo->siguiente;
			delete nodo;
		   }

	}else{
		cout<<"\nDato "<<n<<" no encontrado.\nNo se pudo eliminar"<<endl;
	}
  }else cout<<"\nLista Vacia"<<endl;
}
```

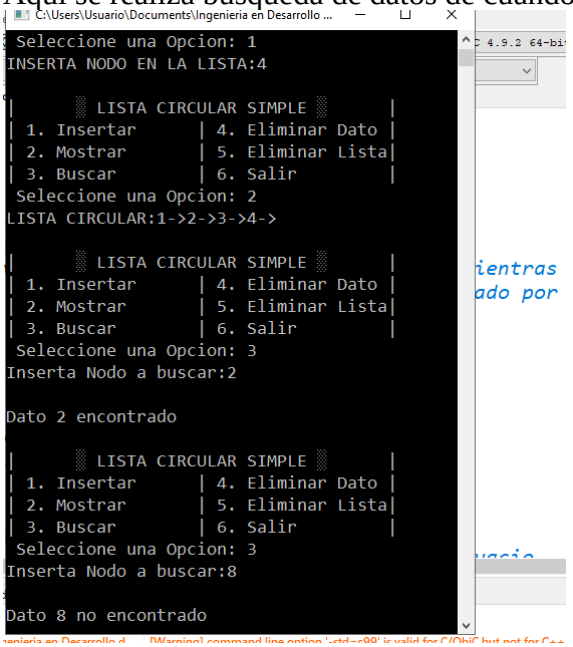
```
int main(){
	int opc,valor;
	Lista lista;
	do{
		cout << "\n|      ° LISTA CIRCULAR SIMPLE °      |";
		cout << "\n| 1. Insertar      | 4. Eliminar Dato |";
		cout << "\n| 2. Mostrar       | 5. Eliminar Lista|";
		cout << "\n| 3. Buscar        | 6. Salir         |";
		cout << "\n Seleccione una Opcion: ";
		cin >> opc;
		switch(opc){
		case 1:
			cout << "INSERTA NODO EN LA LISTA:";
			cin>>valor;
			lista.insertarNodo(valor);
			break;
		case 2:
			cout << "LISTA CIRCULAR:";
			lista.mostrarLista();
			break;
		case 3:
			cout << "Inserta Nodo a buscar:";
			cin>>valor;
			lista.buscarNodo(valor);
			break;
		case 4:
			cout << "Inserta Nodo a eliminar:";
			cin>>valor;
			lista.borrarNodo(valor);
			break;
		case 5:
			cout << "ELIMINAR LISTA CIRCULAR";
			lista.~Lista();
			cout<<"\nLISTA BORRADA"<<endl;
			break;

		case 6: break;
		default:
			cout << "\n\n Opcion No Valida \n\n";
		}
	} while (opc != 6);

	return 0;
}
```

-Resultados

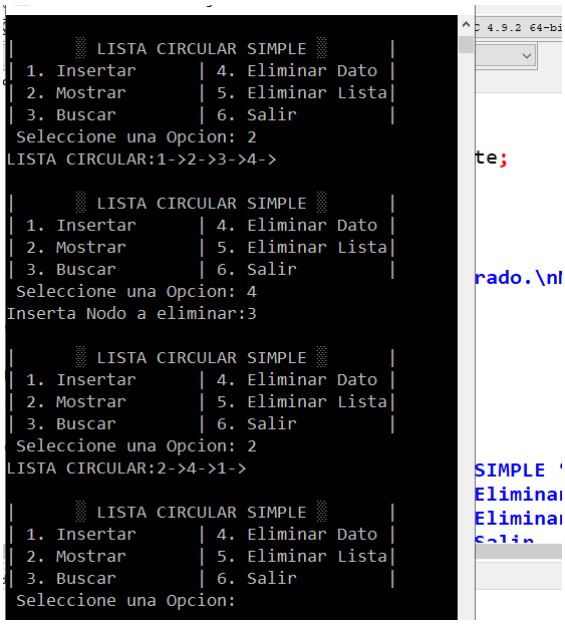
Aquí se realiza la inserción de datos



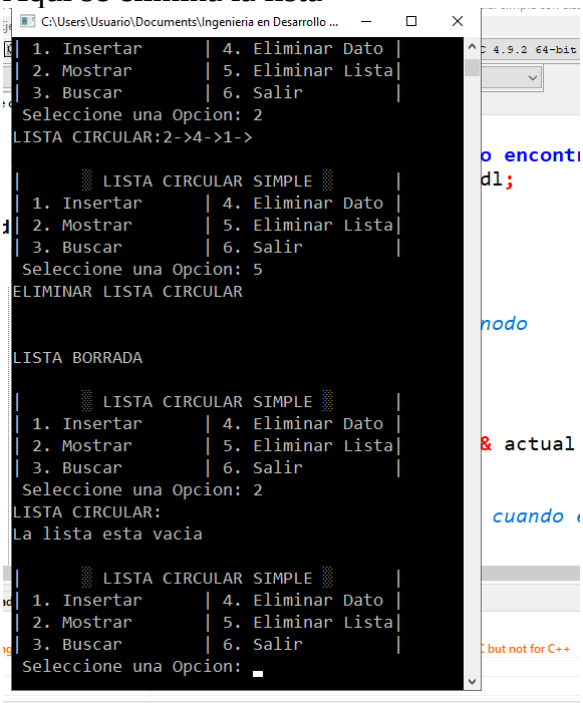
Aquí se realiza búsqueda de datos de cuando se encuentra al dato y cuando no.



Aquí se realiza la eliminación de un nodo



Aquí se elimina la lista



-Conclusión

Esta práctica me pareció muy interesante ya que aquí las listas circulares son diferentes a las listas simplemente enlazadas ya que aquí nunca apunta a null al final, sino que apunta de nuevo al inicio de la lista por lo que la lista nunca acaba y se tiene que realizar otras condicionales para insertar, buscar y eiliminar un nodo. Me ha agradado realizar estas practicas de realizar un programa de estrucuturada a orientada a objetos porque asi he podido estar practicando más mi programación orientada objetos.

-Referencias

Estructura de Datos. (s. f.). *Listas Circulares*. Recuperado 24 de octubre de 2021, de http://estr-datos-omar.blogspot.com/2011/10/listas-circulares.html

Moisset, D. (s. f.). *Estructuras dinámicas en C++: Listas genéricas circulares*. C con Clase. Recuperado 24 de octubre de 2021, de https://www.tutorialesprogramacionya.com/cmasmasya/detalleconcepto.php?codigo=174&punto=43&inicio=30

UAEH. (s. f.). *Listas Circulares*. Recuperado 24 de octubre de 2021, de http://cidecame.uaeh.edu.mx/lcc/mapa/PROYECTO/libro9/listas_circulares.html