Universidad Nacional Autónoma de Nicaragua, León

Facultad de Ciencias y Tecnología

Departamento de Computación

Componente: Aplicaciones de Estructuras de Datos

Guía #8:

Título de la Guía: LISTAS CIRCULARES.

Objetivos:

- Construir programas utilizando listas circulares como tipo dinámico de datos y resolver problemas de la vida cotidiana por medio de sus operaciones.

Temporización:

- Noviembre, 2020

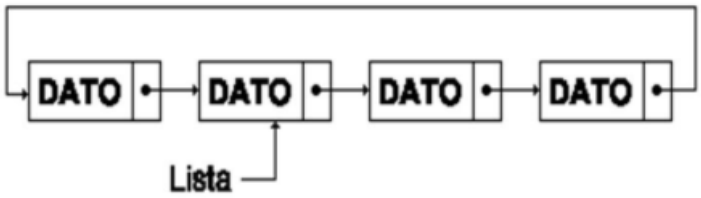
Elaborado por:

- Ing. David Maradiaga Gutiérrez

"A la libertad por la Universidad"

LISTAS CIRCULARES

Una lista circular es una lista lineal en la que el último nodo a punta al primero. Las listas circulares evitan excepciones en las operaciones que se realicen sobre ellas. No existen casos especiales, cada nodo siempre tiene uno anterior y uno siguiente. En algunas listas circulares se añade un nodo especial de cabecera, de ese modo se evita la única excepción posible, la de que la lista esté vacía.



Los tipos que definiremos normalmente para manejar listas cerradas son los mismos que para manejar listas abiertas:

```
typedef struct _nodo

{
        int dato;
        struct _nodo *siguiente;
} tipoNodo;


typedef tipoNodo *pNodo;

typedef tipoNodo *Lista;
```

1. Ejercicios Resueltos

1.1. Programa que permita crear una lista circular y pone a prueba las operaciones básicas.

```
//Lista circulares
#include <stdlib.h>
#include <stdio.h>
typedef struct _nodo
{
        int valor;
        struct _nodo *siguiente;
} tipoNodo;
```

```
typedef tipoNodo *pNodo;
typedef tipoNodo *Lista;

// Funciones con listas:
void Insertar(Lista *l, int v);
void Borrar(Lista *l, int v);
void BorrarLista(Lista *);
void MostrarLista(Lista l);

int main()
{
	Lista lista = NULL;
	//pNodo p;

	Insertar(&lista, 10);
	printf("Insertar 10\n");
	Insertar(&lista, 40);
	printf("Insertar 40\n");
	Insertar(&lista, 30);
	printf("Insertar 30\n");
	Insertar(&lista, 20);
	printf("Insertar 20\n");
	Insertar(&lista, 50);
	printf("Insertar 50\n");
	MostrarLista(lista);

	Borrar(&lista, 30);
	printf("\nBorrar 30\n");
	Borrar(&lista, 50);
	printf("Borrar 50\n");
	MostrarLista(lista);

	printf("\nBorrando lista\n");
	BorrarLista(&lista);

	system("PAUSE");
	return 0;

}
void Insertar(Lista *lista, int v)
{
	pNodo nodo;

	// Creamos un nodo para el nuevo valor a insertar
```

```
        nodo = (pNodo)malloc(sizeof(tipoNodo));
        nodo->valor = v;

        // Si la lista está vacía, la lista será el nuevo nodo
        // Si no lo está, insertamos el nuevo nodo a continuación del apuntado
        // por lista
        if(*lista == NULL)
                *lista = nodo;
        else nodo->siguiente = (*lista)->siguiente;
        // En cualquier caso, cerramos la lista circular
        (*lista)->siguiente = nodo;
}

void Borrar(Lista *lista, int v)
{
        pNodo nodo;
        nodo = *lista;
        // Hacer que lista apunte al nodo anterior al de valor v
        do
        {
                if((*lista)->siguiente->valor != v)
                        *lista = (*lista)->siguiente;
        } while((*lista)->siguiente->valor != v && *lista != nodo);

        // Si existe un nodo con el valor v:
        if((*lista)->siguiente->valor == v)
        {       // Y si la lista sólo tiene un nodo
                if(*lista == (*lista)->siguiente)
                {
                        // Borrar toda la lista
                        free(*lista);
                        *lista = NULL;
                }
                else {
                        // Si la lista tiene más de un nodo, borrar el nodo  de valor v
                        nodo = (*lista)->siguiente;
                        (*lista)->siguiente = nodo->siguiente;
                        free(nodo);
                }
        }
}

void BorrarLista(Lista *lista)
{
```

```
	pNodo nodo;

	// Mientras la lista tenga más de un nodo
	while((*lista)->siguiente != *lista)
	{
		// Borrar el nodo siguiente al apuntado por lista
		nodo = (*lista)->siguiente;
		(*lista)->siguiente = nodo->siguiente;
		free(nodo);
	}

	// Y borrar el último nodo
	free(*lista);
	*lista = NULL;
}

void MostrarLista(Lista lista)
{
	pNodo nodo = lista;

	printf("\n>Mostrar lista:\n");
	do
	{
		printf("%d -> ", nodo->valor);
		nodo = nodo->siguiente;
	} while(nodo != lista);

	printf("\n");
}
```

Salida de ejecución:

```
Insertar 10
Insertar 40
Insertar 30
Insertar 20
Insertar 50

>Mostrar lista:
10 -> 50 -> 20 -> 30 -> 40 ->

Borrar 30
Borrar 50

>Mostrar lista:
10 -> 20 -> 40 ->

Borrando lista
Presione una tecla para continuar . . .
```

1.2. Modificar el ejemplo anterior creando un menú de opciones para el usuario y mejorar así la interacción con la aplicación.

```
//Lista circulares
#include <stdlib.h>
#include <stdio.h>
typedef struct _nodo
{
	int valor;
	struct _nodo *siguiente;
} tipoNodo;

typedef tipoNodo *pNodo;
typedef tipoNodo *Lista;

// Funciones con listas:
void Insertar(Lista *l, int v);
void Borrar(Lista *l, int v);
void BorrarLista(Lista *);
void MostrarLista(Lista l);

int main()
{
	Lista lista = NULL;
	//pNodo p;
	int v, op;

	for(;;)
	{
		printf("\n\tMENU DE OPERACIONES\n");
		printf("1. Insetar nodo.\n");
		printf("2. Mostrar Lista.\n");
		printf("3. Borrar nodo.\n");
		printf("4. Salir.\n");
		printf("opcion: ");
		scanf("%d", &op);

		switch(op)
		{
		case 1:
			printf("\nValor a insertar: ");
			scanf("%d", &v);

			Insertar(&lista, v);
			break;

		case 2:
			MostrarLista(lista);
			break;
```

```
                case 3:
                        printf("\nValor a borrar: ");
                        scanf("%d", &v);

                        Borrar(&lista, v);
                        break;

                case 4:
                        BorrarLista(&lista);
                        exit(-1);

                default:
                        printf("\nOpcion invalida!\n");
                }
        }

        return 0;

}
void Insertar(Lista *lista, int v)
{
        pNodo nodo;

        // Creamos un nodo para el nuvo valor a insertar
        nodo = (pNodo)malloc(sizeof(tipoNodo));
        nodo->valor = v;

        // Si la lista está vacía, la lista será el nuevo nodo
        // Si no lo está, insertamos el nuevo nodo a continuación del apuntado
        // por lista
        if(*lista == NULL)
                *lista = nodo;
        else nodo->siguiente = (*lista)->siguiente;
        // En cualquier caso, cerramos la lista circular
        (*lista)->siguiente = nodo;
}

void Borrar(Lista *lista, int v)
{
        pNodo nodo;
        nodo = *lista;

        if(!nodo)
        {
                printf("\n> Lista vacia!\n");
                return;
        }

        // Hacer que lista apunte al nodo anterior al de valor v
```

```
	do
	{
		if((*lista)->siguiente->valor != v)
			*lista = (*lista)->siguiente;
	} while((*lista)->siguiente->valor != v && *lista != nodo);

	// Si existe un nodo con el valor v:
	if((*lista)->siguiente->valor == v)
	{	// Y si la lista sólo tiene un nodo
		if(*lista == (*lista)->siguiente)
		{
			// Borrar toda la lista
			free(*lista);
			*lista = NULL;
		}
		else {
			// Si la lista tiene más de un nodo, borrar el nodo  de valor v
			nodo = (*lista)->siguiente;
			(*lista)->siguiente = nodo->siguiente;
			free(nodo);
		}
	}
}

void BorrarLista(Lista *lista)
{
	pNodo nodo;

	if(!(*lista))
	{
		printf("\n> Lista vacia!\n");
		return;
	}

	// Mientras la lista tenga más de un nodo
	while((*lista)->siguiente != *lista)
	{
		// Borrar el nodo siguiente al apuntado por lista
		nodo = (*lista)->siguiente;
		(*lista)->siguiente = nodo->siguiente;
		free(nodo);
	}

	// Y borrar el último nodo
	free(*lista);
	*lista = NULL;
}

void MostrarLista(Lista lista)
{
```

```
	pNodo nodo = lista;

	printf("\n>Mostrar lista:\n");

	if(!nodo)
	{
		printf("\n> Lista vacia!\n");
		return;
	}

	do
	{
		printf("%d -> ", nodo->valor);
		nodo = nodo->siguiente;
	} while(nodo != lista);

	printf("\n");
}
```

Salida de ejecucion:

```
        MENU DE OPERACIONES
1. Insetar nodo.
2. Mostrar Lista.
3. Borrar nodo.
4. Salir.
opcion: 1

Valor a insertar: 20

        MENU DE OPERACIONES
1. Insetar nodo.
2. Mostrar Lista.
3. Borrar nodo.
4. Salir.
opcion: 1

Valor a insertar: 40

        MENU DE OPERACIONES
1. Insetar nodo.
2. Mostrar Lista.
3. Borrar nodo.
4. Salir.
opcion: 2

>Mostrar lista:
20 -> 40 ->

        MENU DE OPERACIONES
1. Insetar nodo.
2. Mostrar Lista.
3. Borrar nodo.
4. Salir.
opcion:
```

2. Ejercicios propuestos

2.1. Ejercicios Propuesto 1:

Modifique el ejercicio propuesto #3 de la guía Listas doblemente enlazadas, de tal forma que aplique su funcionalidad por medio de una Lista circular.

2.2. Ejercicios Propuesto 2:

En un hotel se tiene almacenada información sobre las habitaciones y huéspedes del mismo. Diseñe un programa que permita crear una lista circular clasificada para almacenar la información, en la cual cada elemento conste de los siguientes campos: **NumHab** (número de habitación), **Nombre** (nombre del cliente), **FechaArr** (fecha de arribo del cliente: fecha actual del sistema), **Precio** (precio de la habitación) y un puntero a un elemento del mismo tipo.

La información de la lista estará ordenada teniendo en cuenta el número de la habitación. Modifique la función de inserción vista en clases para tal fin.

El programa incluirá las siguientes operaciones:

a) Registrar un huésped.

b) Borrar un huésped de la lista.

c) Buscar un huésped en la lista.

d) Visualizar un informe de los huéspedes registrados en la lista (formato tabular).

e) Facturación: dado el nombre de un huésped y la fecha actual, regrese cuanto debe pagar dicho huésped (asuma que la diferencia entre las fechas la da el número de días de alojamientos).

3. Bibliografía

- Ceballos Francisco Javier. **Curso de Programación C/C++**. Segunda Edición. Editorial RAMA, Madrid.

- Joyanes Aguilar Luis, Zahonero Martínez Ignacio. **Programación en C. Metodología, estructura de datos y objetos**. Mc Graw Hill.