	Carátula para entrega de prácticas
Facultad de Ingeniería	Laboratorio de docencia

Laboratorios de computación salas A y B

Profesor:	Ing. Adrian Ulises Martínez Mercado
Asignatura:	Algoritmos y Estructura de Datos I
Grupo:	13
No de Práctica(s):	12
Integrante(s):	Hernández Rojas Mara Alexandra
No. de Equipo de cómputo empleado:	No Aplica
No. de Lista o Brigada:	7
Semestre:	2020-2
Fecha de entrega:	07/06/2020
Observaciones:	

CALIFICACIÓN: _____

Introducción

El objetivo de la práctica es diseñar logaritmos recursivos para resolver problemas.

Desarrollo con ejercicios

Actividad 1

Crearemos una lista doblemente ligada que elimine de manera recursiva todos los nodos en la lista.

```h
/*
 * Programador: Hernandez Rojas Mara Alexandra Practica 12
 * Este programa define las estructuras de nodo y lista*/
#ifndef E1_H
#define E1_H

//typedef struct _info INFO;

typedef struct _info{
    char nombre[32];
    char apellido[64];
} INFO; //info1,info2, info3;

typedef struct _node NODE;

struct _node{
    INFO info;
    NODE *next;
    NODE *prev;
};

//typedef struct _list LIST;

typedef struct _list {
    NODE *tail;
    NODE *head;
}LIST;


void insertar(INFO info, LIST *l);
LIST *crear_lista();
void eliminar(LIST *l);
void imprimir(LIST *l);

NODE *crear_nodo();
void borrar_nodos(NODE *n);

#endif
//NODE y LIST
~
```

e1.h

```c
/*
 * Programador: Hernandez Rojas Mara Alexandra Practica 12
 * Este programa define las estructuras de nodo y lista*/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "e1.h"

void insertar(INFO info, LIST *l){
    if(l!=NULL){
        if(l->head == NULL){
            l->head = crear_nodo();
            l->head->info = info;
            return;
        }else{
                NODE *nuevo = crear_nodo();
                //strcpy(nuevo->info.nombre , info.nombre);
                //strcpy(nuevo->info.apellido, info.apellido);
                nuevo->info = info;
                nuevo->next = l->head;
                l->head->prev = nuevo;
                l->head = nuevo;
                }
        }
}

LIST *crear_lista(){
    LIST *l =(LIST*) malloc(sizeof(LIST));
    l->head = NULL;
    l->tail = NULL;
    return l;
}

void eliminar(LIST *l){
    if(l->head!=NULL){
        borrar_nodos(l->head);
    }else{
            free(l);
    }
}
```

e1.c (P1)

```c
NODE *crear_nodo(){
    NODE *n =(NODE*) malloc(sizeof(NODE));
    n->next = NULL;
    n->prev = NULL;
    strcpy(n->info.nombre , " ");
    strcpy(n->info.apellido , " ");
    return n;
}

void borrar_nodos(NODE *n){
    if (n->next!=NULL ){       //Caso recursivo
        borrar_nodos(n->next);
    }else{
        n->prev= NULL;            //Caso base
        free(n);
    }
}

void imprimir(LIST *l){
   for(NODE *i = l->head; i!=NULL; i = i->next){
        printf("%s, %s\n", i->info.nombre, i->info.apellido);
    }
}

/*Vamos a borrar los ultimos nodos hasta que ya no queda ninguno
 *Caso base sirve para darle fin a las fnciones recursivas
 *si no hay caso base nunca va a parar las llamadas recursivas
 *NO sólo hay errores del código hay errores de lógica del programa
 *Al momento de insetar funciona un poco como una pila
 */
```

e1.c (P2)

```c
/*
 * Programador: Hernandez Rojas Mara Alexandra Practica 12
 * Este programa define las estructuras de nodo y lista*/

#include <stdio.h>
#include<string.h>
#include "e1.h"

int main(){
    LIST *lista;
    INFO info1, info2, info3;

    lista = crear_lista();

    strcpy(info1.nombre, "Juan");
    strcpy(info1.apellido, "Perez");
    insertar(info1, lista);
    //imprimir(lista);

    strcpy(info2.nombre, "Margarita");
    strcpy(info2.apellido, "Lopez");
    insertar(info2, lista);
    //imprimir(lista);

    strcpy(info3.nombre, "Laura");
    strcpy(info3.apellido, "Martinez");
    insertar(info3, lista);
    //imprimir(lista);

    imprimir(lista);
    eliminar(lista);

    return 0;
}
```

main.c

```
019-2020/2020-2/EDA/7JUNIO/Practica12
$ ./a
Laura, Martinez
Margarita, Lopez
Juan, Perez
```

ejecución en consola

Caso recursivo: El siguiente del nodo apuntado es diferente de NULO
Caso base: El siguiente del nodo apuntado es NULO
Operacion: borrar_nodos(NODE *n);

Actividad 2

Aquí programamos una clase lista con programación orientada a objetos, se parecen a las estructuras:
Clase: Nodo contiene un nombre, un apellido, y un marcador de clase Nodo a siguiente vacío
Clase: Lista contiene un marcador de clase Lista a head vacío y un marcador de clase Lista a tail vacío

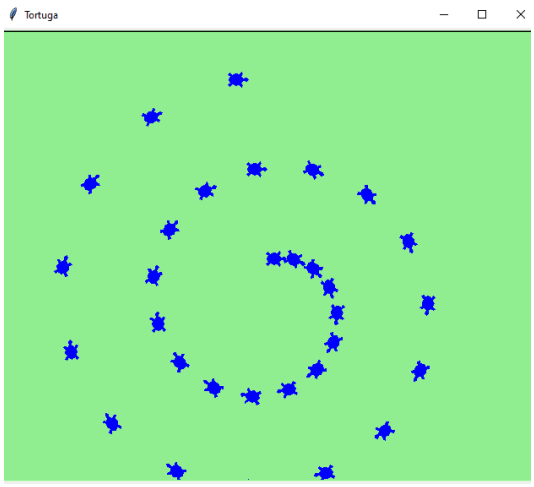
Operaciones de la clase Lista: Agrega nodos a la lista y Elimina nodos de la lista

```
"""
Progrmador Hernandez Rojas Mara Alexandra Practica 12
Pila con programacion orientada a objetos en Python

En programación orientada a objetos las clases se parecen a las estructuras no s
"""
class Nodo:
    def __init__(self):
        self.nombre = None
        self.apellidos = None
        self.next = None

    def eliminar(self):
        self = None


class Lista:
    def __init__(self):
        self.head = None
        self.tail = None

    def agregar(self, nombre, apellidos):
        if self.head == None:
            self.head = Nodo()#Aqui mandas llamar al init de nodo
            self.head.nombre = nombre
            self.head.apellidos = apellidos
            return
        temporal = Nodo()
        temporal.nombre = nombre
        temporal.apellidos = apellidos
        temporal.next = self.head
        self.head = temporal

    def eliminar(self):
        temp = self.head
        while temp.next != None:
            while temp.next != None:
                temp.eliminar()
        temp.eliminar()
"""
El self es como un malloc reserva memorias asocia a las clases con su miembros
struct nodo{
    char *nombre;       Y una fnción:
    char *apellidos;        nodo *crear_nodo();
    nodo *next
```

Actividad 3

Aquí programamos Una gráfica que muestra una Tortuga con el recorrido iterativo en un ciclo for:

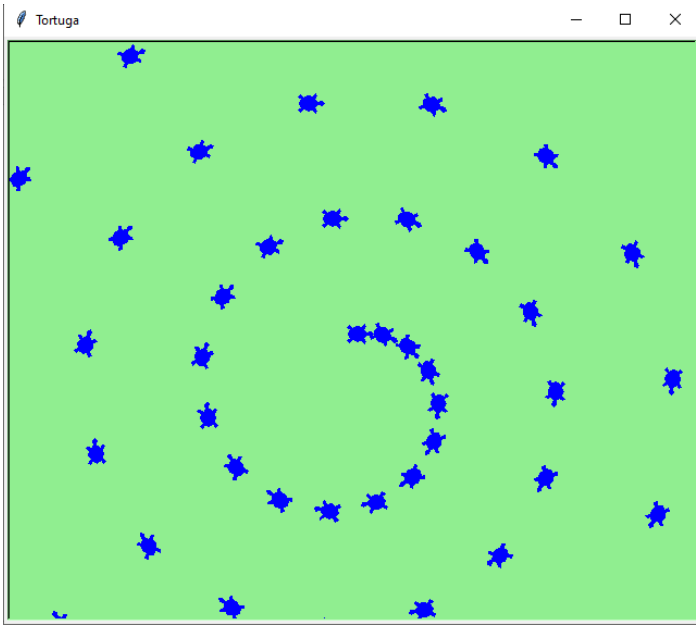
```
"""
Programador Hernandez Rojas Mara Alexandra Practica 12
Grafica Tortugas forma iterativa
"""

import turtle

wn = turtle.Screen()
wn.bgcolor("lightgreen")
wn.title("Tortuga")
tess = turtle.Turtle()
tess.shape("turtle")
tess.color("blue")
tess.penup()
size = 20

for i in range(30):
    tess.stamp()
    size = size+ 3
    tess.forward(size)
    tess.right(24)

wn.mainloop()
```



Actividad 4

Aquí programamos la misma gráfica que muestra una Tortuga pero esta vez con un recorrido recursivo y pasando argumentos desde línea de commandos `Python e4.py --huella 45`
Caso Recursivo Huella > Cero
Caso Base: Huella = Cero
Operación Base recorrido_recursvo(Tortuga, espacio, huella)

```
"""
Programador Hernandez Rojas Mara Alexandra Practica 12
Grafica Tortugas forma recursiva
"""
import turtle
import argparse

def recorrido_recursivo(tortuga, espacio, huella):
    if huella >0:
        tortuga.stamp()
        espacio = espacio + 3
        tortuga.forward(espacio)
        tortuga.right(24)
        recorrido_recursivo(tortuga, espacio, huella-1)

ap = argparse.ArgumentParser()
ap.add_argument("--huella",required=True, help="numero de huellas")
args = vars(ap.parse_args())
huellas = int(args["huella"])

wn = turtle.Screen()
wn.bgcolor("lightgreen")
wn.title("Tortuga")
tess = turtle.Turtle()
tess.shape("turtle")
tess.color("blue")

tess.penup()
recorrido_recursivo(tess, 20, huellas)

wn.mainloop()

"""
Cuando llegue a uno pinta la primera huella
Caso recursivo Huellas = 0
Operación Dibujar Tortuga
Caso
```



Conclusiones

Tanto en lenguaje C como en Python implementar una función recursiva es fácil si tienes en mente tres cosas escenciales primero **la operación base** que es la acción que quieres que se repita, **el caso recursivo** que son las condiciones que se tienen que cumplir para que la operación básica se repita y **el caso base** que son las condiciones en lalas que ya no es necesario ejecutar la operación base de forma recursiva.

La ventaja de la recursividad es que ahorra muchas líneas de código, una opercación complicada como lo era borrar los nodos de una lista doblemente ligada llevaba cerca de 15 líneas de código en programas anteriores haciéndolo de forma iterative mientras que al definirla con un algoritmo recursivo quedó definida tan solo con 5 líneas.

La desventaja de las operaciones recursivas es lque la cantidad de recursos que consumen supera con creces los recursos que consume la forma iterativa. Claro que resulta mucho más práctico recurrir a la recursividad cuando la forma de definer una operación mediante iteraciones se Vuelve excesivamente complicada.