	Carátula para entrega de prácticas
Facultad de Ingeniería	Laboratorio de docencia

Laboratorios de computación salas A y B

Profesor: Adrian Ulises Mercado Marínez

Asignatura: Estructura de Datos y Algoritmos I

Grupo: 13

No de Práctica(s): 12

Integrante(s): Martínez Salgado Elean Jim

No. de Equipo de cómputo empleado: -

No. de Equipo de cómputo empleado:

No. de Lista o Brigada:

Semestre: 2020-2

Fecha de entrega: 7 de junio de 2020

Observaciones:

CALIFICACIÓN: _____

Introducción

la recursión es básicamente resolver un problema mediante la resolución de otros mas pequeños, en esta practica se hará énfasis en este método, comparándolo con la resolución de un problema sin recurrir a la recursividad.

Desarrollo

1- Creamos una lista con la función de borrar recursiva, usando el caso n->prev es igual a nulo

eje.h

```c
#ifndef E1_H
#define E1_H

typedef struct _node NODE;

typedef struct _info{
    char nombre[32];
    char apellido[64];

} INFO;

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

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
```

eje1.c

```c
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include "ejercicio.h"

void insertar(INFO info, LIST *l){
    if(l!=NULL){
        if(l->head==NULL){
            l->head = crear_nodo();
            l->head->info = info;
            return;
        }
        NODE *nuevo = crear_nodo();
        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);
    }
    free(l);
}
```

```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);
    }
    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);
    }
}
```

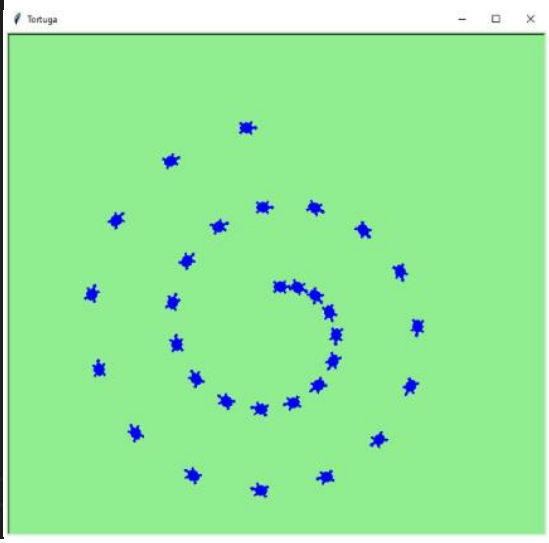
main.c

```c
#include<stdio.h>
#include "ejercicio.h"
#include <string.h>

int main(){
    LIST *lista;
    INFO info;
    strcpy(info.nombre, "nombre1");
    strcpy(info.apellido, "apellido11 apellido12");

    lista = crear_lista();
    insertar(info,lista);
    imprimir(lista);
    eliminar(lista);
    return 0;
}
```

2- Recorrido de la tortuga sin recursividad

2.py

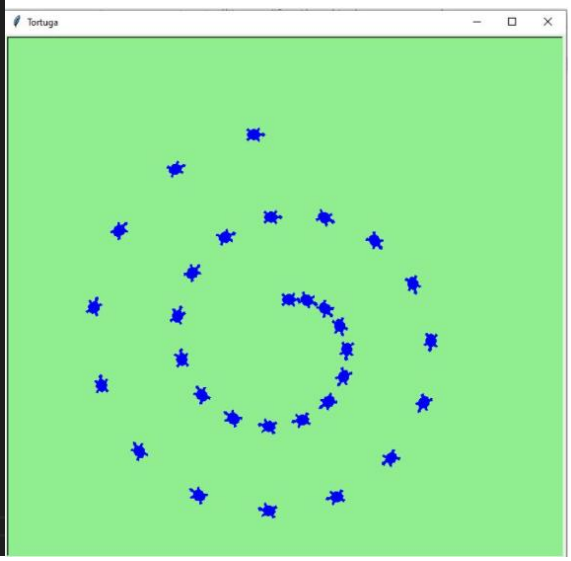
```
1   import turtle
2
3   wn = turtle.Screen()
4   wn.bgcolor("lightgreen")
5   wn.title("Tortuga")
6   tess = turtle.Turtle()
7   tess.shape("turtle")
8   tess.color("blue")
9
10  tess.penup()
11  size = 20
12  for i in range(30):
13      tess.stamp()
14      size = size+3
15      tess.forward(size)
16      tess.right(24)
17
18  wn.mainloop()
```



3- Mismo ejercicio pero con recursividad

3.py

```python
import turtle
import argparse

def recorrido_recursivo(tortuga, espacio, huellas):
    if huellas > 0:
        tortuga.stamp()
        espacio = espacio + 3
        tortuga.forward(espacio)
        tortuga.right(24)
        recorrido_recursivo(tortuga, espacio, huellas-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, 20,30)
wn.mainloop()
```



Conclusiones

La recursividad es una herramienta muy útil, ya que puede ser útil para reducir el tamaño de algunos algoritmos, aunque no es precisamente sencilla su implementación.