	Carátula para entrega de prácticas
Facultad de Ingeniería	Laboratorio de docencia

Laboratorios de computación salas A y B

Profesor: Adrian Ulises Mercado

Asignatura: Estructura de datos y Algoritmos I

Grupo: 13

No de Práctica(s): Práctica 12

Integrante(s): José Abraham Hernández Vargas

No. de Equipo de cómputo empleado:

No. de Lista o Brigada: Brigada 5

Semestre: 2020-2

Fecha de entrega: 7-06.2020

Observaciones:

CALIFICACIÓN: _____

INTRODUCCIÓN

Se van a desarrollar algunos programas con funciones recursivas, para tener noción de cómo implementarlas en futuras prácticas o proyectos y cuales son sus implicaciones

DESARROLLO

Recursividad en c

Se desarrolla una lista doblemente ligada, en la cual se van a guardar los nombres y los apellidos de una persona, para ello se desarrolla una estructura la cual se va a llamar INFO, una vez que establecimos la estructura, se desarrolla la estructura del nodo, el cual contiene una variable info de tipo INFO; Una vez que se establece una variable info, se establecen los apuntadores next y prev de tipo NODE. Para finalizar la lista doblemente ligada se crea una estructura denominada LIST, la cual está compuesta de de dos apuntadores: head y tail. Finalmente se escribieron los prototipos de las funciones que va a realizar la lista.

list.h

```
#ifndef E1_H
#define E1_h

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

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

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

LIST *crear_list();
void insertar(INFO info, LIST *l);
void eliminar(LIST *l);
NODE *crear_nodo();
void borrar_nodos(NODE *n);
void imprimir (LIST *l);
#endif
```

List.c

La operación que nos interesa es la de borrar nodos, ya que ésta operación es recursiva. El objetivo de la función borrar nodos, como su nombre lo indica es borrar todos los nodos de una lista. Sabemos que ésta función es recursiva porque dentro de la condición if, se vuelve a llamar la función, donde su caso base es que el previo de un nodo sea nulo.

```c
#include<stdio.h>
#include<stdlib.h>
#include "e1.h"

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

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;
    }
}

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

NODE *crear_nodo(){
    NODE *n = (NODE*) malloc(sizeof(NODE));
    n->prev = NULL;
    n->next = NULL;
    return n;
}

void borrar_nodos(NODE *n){
    if(n->next != NULL){
        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

Finalmente para ejecutar el código, se crea un archivo main en donde se incluyen los archivos .h y .c anteriormente descritos.

En función main se establece que se van a introducir 3 nombre, por lo que se utiliza la función strcpy de la biblioteca string, para copiar la cadena de caracteres y guardarla en info[i]. nombre o info[i].apellido, para que posteriormente se imprima la lista.

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


int main(){
    LIST *lista;
    INFO info[3];


    strcpy(info[0].nombre, "Darrion");
    strcpy(info[0].apellido, "Rohan Hagenes");
    strcpy(info[1].nombre, "Elian");
    strcpy(info[1].apellido, "Brenna Langworth");
    strcpy(info[2].nombre, "Shania");
    strcpy(info[2].apellido, "Carmella Gaylord");
    lista = crear_list();
    insertar(info[0], lista);
    insertar(info[1], lista);
    insertar(info[2], lista);
    imprimir(lista);
    eliminar(lista);
    return 0;
}
```

Recursividad en python

Se importa la biblioteca turtle para poder visualizar los pasos que va a realizar la tortuga.

Por otro lado también se va a importar la biblioteca `argparse`, que nos permite mandar datos de entrada por medio de banderas.

Se desarrolla una función def denominada recorrido recursivo donde se ponen las características de los movimientos que va realizar la tortuga como:

tortuga.forward(espacio) : el cual nos indica cómo queremos que se mueva: hacia adelante o hacia atrás.

tortuga.right(24) : Este parámetro nos indica si la tortuga abre hacia la izquierda o la derecha.

tortuga.stamo( ): Huella de la tortuga
size = size +3 : La forma en cómo queremos que se incremente el paso de la tortuga.

Como vamos a ingresar el número de huellas que realice la tortuga por líneas de comando, se utiliza la biblioteca argparse, la cual se va a encargar de transformar el string ”huellas” en un entero.
Para ello, se inicializa la variable ap con la función argparse.ArgumentParser().

Después se agrega el dato de entrada con la bandera --huellas, también establecemos que se debe ingresar un número, con requiered y un aviso que escriba el “número de huellas“ por si no se sabe que se debe escribir

ap.add_argument(“--huellas”, requiered = true, help = “número de huellas”)

Una vez que se tiene el dato de entrada, se convierte en un valor, ya que ap.argparse sigue siendo un string.
args = vars(ap.argparse( ))
Finalmente se iguala el número de huellas que va a realizar la tortuga con el valor de entrada:
huellas = int(args[“huellas”].

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

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()
```

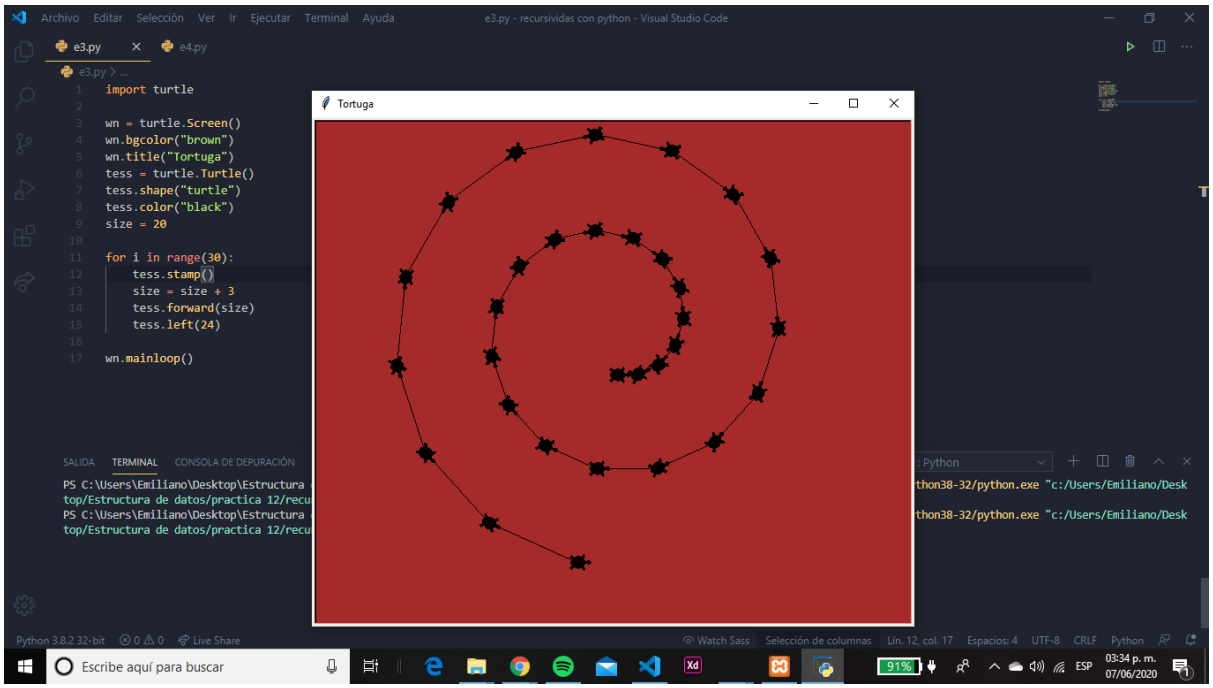
EJECUCIÓN

Ejecución en c

```
SALIDA   TERMINAL   CONSOLA DE DEPURACIÓN   PROBLEMAS
PS C:\Users\Emiliano\Desktop\Estructura de datos\practica 12\recursividad> ./main
nombre1, apellido 1 apellido 2
PS C:\Users\Emiliano\Desktop\Estructura de datos\practica 12\recursividad> ./main
nombre1, apellido 1 apellido 2
PS C:\Users\Emiliano\Desktop\Estructura de datos\practica 12\recursividad> gcc *.c -o main
PS C:\Users\Emiliano\Desktop\Estructura de datos\practica 12\recursividad> ./main
Shania, Carmella Gaylord
Elian, Brenna Langworth
Darrion, Rohan Hagenes
PS C:\Users\Emiliano\Desktop\Estructura de datos\practica 12\recursividad>
Selección de columnas   2 selecciones (1 caracte
```

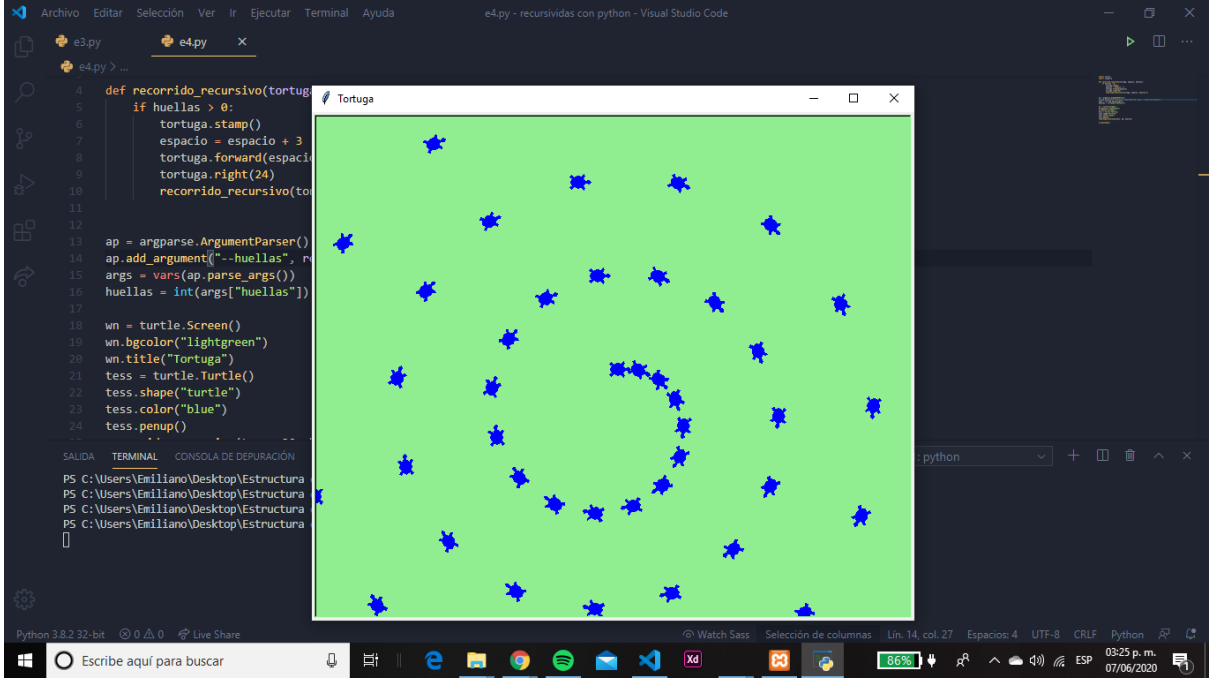
Ejecución en python

e3.py (Ejercicio no recursivo)



e4.py(Ejercicio recursivo)

para ejecutar el ejercicio recursivo, es necesario que en la terminal se escriba el nombre del archivo, seguido de --huellas y el número de pasos que quieres que realice la tortuga.
En este caso la tortuga realizó 45 pasos



CONCLUSIÓN

La recursividad en programación es una gran herramienta pues te ahorras varias líneas de código y puede quedar más entendible la función de esta

COMENTARIOS

Abraham: Se me hizo un poco confuso las palabras reservadas para los algoritmos con turtle pero el resultado de la ejecución me parece fantástico.

BIBLIOGRAFÍAS