Universidad de Costa Rica
Facultad de Ingeniería
Escuela de Ingeniería Eléctrica

IE-0117 Programación Bajo Plataformas Abiertas

MSc. Andrés Mora Zúñiga - I Ciclo 2020

Laboratorio # 6

C: Structs, punteros y memoria dinámica

Instrucciones Generales:
Los laboratorios se deben realizar de manera individual.
El laboratorio debe entregarse antes del 19 de Junio a las 23:59.
Entregue un archivo comprimido que incluya un directorio llamado `informe` con los archivos necesarios para generar el PDF del informe (.tex, imágenes, código, entre otros) y un directorio `src` con los archivos `.h`, y `.c` que lleven a la solución de cada ejercicio. Cualquier otro formato o entrega tardía no se revisará y el laboratorio tendrá una nota de cero.

1. Introducción

El siguiente laboratorio tiene como objetivo comprobar los conocimientos adquiridos durante el curso. Específicamente se evaluará el buen manejo del lenguaje de programación C. Los temas a evaluar son: tipos de datos personalizados, funciones, manipulación de structs, punteros, memoria dinámica.

Para ello se presente como ejercicio académico la implementación de un programa en C capaz de representar triángulos en un plano cartesiano. Ordenarlos según el tamaño de su área y finalmente imprimirlos en terminal. Además se deberá de automatizar la compilación del programa haciendo uso de un Makefile.

2. Creación de un archivo de encabezado (.h)

Cree un archivo de encabezado llamado `triangulos.h` en el cual deberá incluir todas las bibliotecas (por ejemplo `math.h`) que utilizará. Además debe de definir los tipos de datos `p2D` y `tri`, y declarar los prototipos de las funciones que va a implementar.

2.1. Definición de los tipos de dato p2D y tri. (5 pts)

Defina los siguientes tipos de datos (`typedef`):

- `p2D`: un `struct` que almacena dos `double` (x,y) los cuales representan un punto en el plano cartesiano.
- `tri`: un `struct` que almacena tres `p2D` (A,B,C) los cuales representan un triángulo en el plano cartesiano.

2.2. Prototipos de funciones (5 pts)

Declare los siguientes prototipos de funciones:

- `setSeed`: esta función recibe un `unsigned int` y lo asigna como semilla de la función `srand` incluida desde la biblioteca `stdlib.h`.
- `randNum`: esta función recibe dos `double` (min, max) y regresa un número aleatorio tipo `double` dentro del intervalo definido por $[min, max]$.
- `randP2D`: esta función no recibe parámetros y regresa un `p2D` que representa un punto aleatorio en el plano cartesiano en el intervalo $[-50, 50]$.

- `dist`: esta función recibe 2 parámetros del tipo `p2D` y retorna un `double` con la distancia euclidiana entre los dos puntos.
- `randP2DTri`: esta función recibe como parámetro un `p2D` y regresa un `p2D` aleatorio que se encuentra a una distancia menor a 10 desde el punto recibido como parámetro.
- `triIneq`: esta función recibe como parámetro tres `p2D` y retorna 1 si los tres puntos cumplen desigualdad triangular y 0 si no cumplen desigualdad triangular.
- `randTri`: esta función recibe como parámetro un `p2D` y retorna un `tri`. El `tri` debe cumplir con las siguientes condiciones:
  1. El punto que recibe como parámetro es uno de los vértices del triángulo.
  2. Los otros dos puntos del triángulo se calculan de manera aleatoria, pero deben de estar a una distancia menor a 10 desde el punto definido inicialmente.
  3. Los tres puntos deben cumplir con la desigualdad triangular.
- `calcArea`: esta función recibe como parámetro un `tri` y retorna un `double` que representa el área del triángulo.
- `reserveTri`: esta función recibe como parámetro un `int` que representa la cantidad de `tri` que se desea almacenar. La función retorna un `tri*` que es la dirección de memoria a un bloque de memoria en heap capaz de almacenar dicha cantidad de `tri`.
- `initTri`: esta función recibe como parámetro un `tri*` que contiene la dirección de memoria en la que inicia un bloque de memoria válido para almacenar `tri` y un `int` que representa la cantidad de `tri` que se desea inicializar en dicho bloque. La función procede a crear esa cantidad de `tri` con la función `randTri` y almacenarlos en las direcciones de memoria respectivas en heap.
- `sortTri`: esta función recibe como parámetro un `tri*` que contiene la dirección de memoria en la que inicia el bloque que almacena los `tri` y un `int` que representa la cantidad de `tri` almacenados en el bloque. La función procede a ordenar el bloque de `tri` ascendentemente. Es decir, en la primera posición del bloque queda el *tri* con el área más pequeña y en el último espacio el `tri` con el área más grande.
- `printTri`: esta función recibe como parámetro un `tri` y lo imprime de la siguiente forma:

```
Triangulo:
A: (x1,y1)
B: (x2,y2)
C: (x3,y3)
Area: <area>
```

  Por ejemplo:

```
Triangulo:
A: (0.0, 0.0)
B: (1.0, 0.0)
C: (0.0, 1.0)
Area: 0.5
```

  Preste atención a que los números de punto flotante se imprimen únicamente con un decimal.
- `printAllTri`: esta función recibe como parámetro un `tri*` que contiene la dirección de memoria en la que inicia un bloque que almacena `tri` y un `int` que representa la cantidad de `tri` almacenados en el bloque. La función procede a imprimir cada uno de los triángulos usando la función `printTri`.

3. Creación de un archivo de implementación (.c) (65 pts)

Cree un archivo llamado `triangulos.c` en el cual se incluya el encabezado `triangulos.h` y se implemente el cuerpo de todas las funciones allí contenidas.

4. Programa Principal (15 pts)

El archivo `main.c` debe de realizar lo siguiente:

1. Incluir el encabezado de `triangulos.h` para hacer uso de los tipos de datos creados y todas sus funciones.
2. Recibir por parámetro de línea de comandos la cantidad de triángulos que se desea crear.
3. Crear una variable llamada `bloq` del tipo `tri*` y hacer que apunte a un bloque de memoria en heap que pueda almacenar la cantidad de triángulos especificados por el parámetro de línea de comandos.
4. Llamar a la función `initTri` enviando la variable `bloq` como parámetro para que se llene de triángulos.
5. Llamar a la función `sortTri` enviando la variable `bloq` como parámetro para ordenar los triángulos según el tamaño de su área.
6. Llamar a la función `printAllTri` enviando la variable `bloq` como parámetro para imprimir todos los triángulos almacenados en el bloque.
7. Liberar la memoria reservada en heap por medio del puntero `bloq`.
8. Fin

5. Automatización de la compilación (10 pts)

Finalmente cree un Makefile para este proyecto. El Makefile debe de contener al menos los siguientes targets:

- all: compila los binarios y crea el ejecutable triangulos.x el cual ejecutaría el programa principal.
- clean: elimina todos los archivos temporales e intermedios, así como el ejecutable final.
- run: ejecuta triangulos.x con un argumento de 10.

6. Informe final del laboratorio

Finalmente, luego de concluir con la implementación y automatización de la compilación, debe de realizar un informe técnico con los detalles de la elaboración del código fuente y capturas de pantalla que evidencien la funcionalidad de su código. Si no se presenta el informe escrito se le asignará una nota de cero al laboratorio. Además se debe proveer todo el código fuente generado para resolver el laboratorio y su respectivo Makefile siguiendo el orden solicitado en las instrucciones generales.