Universidad de Costa Rica
Escuela de Ingeniería Eléctrica
Programación Bajo Plataformas Abiertas
MSc. Andrés Mora Zúñiga
I Ciclo 2020
IE-0117

Práctica # 6: C: Structs, punteros y memoria dinámica

Para este laboratorio se implementaron 4 archivos.

1. Triangulos.h

```
#if !defined(FUNCTIONS)
#define FUNCTIONS
#include <stdio.h>
#include <math.h>
#include<stdlib.h>
#include <time.h>
typedef struct p2D
{
    double x;
    double y;

}p2D;

typedef struct tri
{
    p2D A;
    p2D B;
    p2D C;

}tri;

void setSeed(unsigned int semilla);
double randNum(double min, double max);
p2D randP2D();
double dist(p2D p1, p2D p2);
p2D randP2DTri(p2D p1);
int triIneq(p2D p1, p2D p2, p2D p3);
tri randTri(p2D p1);
double calcArea(tri t1);
tri* reserveTri(int cantidadtri);
void initTri(tri* rutamalloc, int cantidadtri);
void sortTri(tri* rutamalloc, int cantidadtri);
void printTri(tri t1);
void printAllTri(tri* rutamalloc, int cantidadtri);
#endif
```

En triangulos.h se incluyen todas las bibliotecas necesarias en este laboratorio. Como la que es usual stdio.h, la que nos ayuda a obtener datos de la linea de comandos stlib.h, la que ayuda a realizar la raíz cuadrada math.h y time.h que es la que ayuda a cambiar la semilla

cada segundo de la función srand. En la linea 7 se puede ver como se define el tipo de dato p2D, al cual se le hace un typedef para indicar que struct p2D= p2D. Un struct p2D se compone de un double x y un double y. En la linea 24 se define el tipo de dato tri. El cual se le hace un typedef para indicar que struct tri=tri. Un struct tri se compone de 3 p2D(A,B,C). Además se puede apreciar que se declaran todas las funciones del archivo triangulo.c

2. Triangulos.c

```
1  #include <stdio.h>
2  #include "triangulos.h"
3  #include <math.h>
4  #include<stdlib.h>
5  #include <time.h>
6
7  void setSeed(unsigned int semilla){
8      srand(semilla);
9  }
10
11 double randNum(double min, double max){
12     int a=(max*10)+1-(min*10);
13     int b=min*10;
14     double numaleatorio =(float)(rand()%(a)+b)/10;
15     return numaleatorio;
16 }
17 p2D randP2D(){
18     int a=500+1+500;
19     int b=-500;
20     int i=0;
21
22     double numaleatorio =(float)(rand()%(a)+b)/10;
23     double numaleatorio2 =(float)(rand()%(a)+b)/10;
24     p2D puntoaleatorio;
25     puntoaleatorio.x = numaleatorio;
26     puntoaleatorio.y = numaleatorio2;
27     return puntoaleatorio;
28 }
29
30 double dist(p2D p1, p2D p2){
31     double xtotal=p1.x-p2.x;
32     double ytotal= p1.y-p2.y;
33     double distancia=sqrt(pow(xtotal,2)+pow(ytotal,2));
34     return distancia;
35 }
36 p2D randP2DTri(p2D p1){
37     p2D puntoaleatorio;
38     double distancia=101;
39     while (distancia>10)
40     {
41         puntoaleatorio=randP2D();
42         distancia=dist(p1,puntoaleatorio);
43     }
44     return puntoaleatorio;
45 }
```

```
46  int triIneq(p2D p1, p2D p2, p2D p3){
47      double lado1=dist(p1,p2);
48      double lado2=dist(p1,p3);
49      double lado3=dist(p3,p2);
50      if ((lado1+lado2)>lado3 && (lado1+lado3)>lado2 && (lado3+lado2)>lado2 )
51      {
52          return 1;
53      }else{
54          return 0;
55      }
56  }
57  tri randTri(p2D p1){
58      p2D p2;
59      p2.x=0;
60      p2.y=0;
61      p2D p3;
62      p3.x=0;
63      p3.y=0;
64      int triIneqq=0;
65      while (triIneqq==0)
66      {
67          while (p2.x==p3.x && p2.y==p3.y){
68              p2=randP2DTri(p1);
69              p3=randP2DTri(p1);
70          }
71          triIneqq=triIneq(p1,p2,p3);
72
73
74      }
75      tri trialeatorio;
76      trialeatorio.A=p1;
77      trialeatorio.B=p2;
78      trialeatorio.C=p3;
79      return trialeatorio;
80  }
81  double calcArea(tri t1){
82      double lado1=dist(t1.A,t1.B);
83      double lado2=dist(t1.A,t1.C);
84      double lado3=dist(t1.C,t1.B);
85      double s= (lado1+lado2+lado3)/2;
86      double area=sqrt(s*(s-lado1)*(s-lado2)*(s-lado3));
87      return area;
88  }
89  tri* reserveTri(int cantidadtri){
90      tri* rutamalloc= malloc(cantidadtri*sizeof(tri));
91      return rutamalloc;
92  }
93  void initTri(tri* rutamalloc, int cantidadtri){
94      for (int i = 0; i < cantidadtri; i++)
95      {
96          p2D puntoaleatorio=randP2D();
97          rutamalloc[i]=randTri(puntoaleatorio);
98      }
99
100 }
```

```
101 void sortTri(tri* rutamalloc, int cantidadtri){
102     tri a;
103     tri b;
104     tri c;
105     int z=0;
106     int k=0;
107     int t=0;
108     tri copiarutamalloc [cantidadtri];
109     for (int i = 0; i < cantidadtri; i++)
110     {
111         copiarutamalloc [i]=rutamalloc[i];
112     }
113     for (int i = 0; i < cantidadtri; i++)
114     {
115         a=copiarutamalloc[i];
116         for (int i = 0; i < cantidadtri; i++)
117         {
118             b=copiarutamalloc[i];
119             if (calcArea(a)>calcArea(b))
120             {
121                 z++;
122             }
123             if (calcArea(a)==calcArea(b))
124             {
125                 if (a.A.x!=b.A.x || a.A.y!=b.A.y || a.B.x!=b.B.x || a.B.y!=b.B
                        .y || a.C.x!=b.C.x || a.C.y!=b.C.y)
126                 {
127                     k++;
128                     if (k>1)
129                     {
130                         t++;
131                         c=b;
132                     }
133 
134                 }
135 
136             }
137         }
138         if (calcArea(a)<calcArea(c)){
139             rutamalloc[z]=a;
140         }else
141         {
142             rutamalloc[z+t]=a;
143         }
144 
145 
146         z=0;
147     }
148 }
149 void printTri(tri t1){
150     printf("Triangulo:\n");
151     printf("A: (%1f,%.1f)\n",t1.A.x,t1.A.y);
152     printf("B: (%1f,%.1f)\n",t1.B.x,t1.B.y);
153     printf("C: (%1f,%.1f)\n",t1.C.x,t1.C.y);
154     printf("Area: %1f\n",calcArea(t1));
```

```
155 }
156 void printAllTri(tri* rutamalloc, int cantidadtri){
157     for (int i = 0; i < cantidadtri; i++)
158     {
159         printTri(rutamalloc[i]);
160     }
161
162 }
```

El archivo triangulos.c contiene el encabezado triangulos.h y se implementa el cuerpo de todas las funciones que se declararon en este.

setSeed: Como se muestra en la linea 7 es una función tipo void que recibe un unsigned int llamado semilla. Este unsigned int es asignado como semilla de la función srand().

randNum: Como se ve en la linea 11 esta función es de tipo double, y recibe por parámetros 2 double de igual manera.Para obtener el numero aleatorio se define un double dentro de la función con etiqueta numaleatorio y se le aplica el modulo a la función rand(). Debido a que es entre los 2 valores de entrada (min y max), para generar un numero aleatorio entre este rango se hizo lo siguiente. Se definieron 2 int debido a que la función con double presentaba errores, donde se declararon a y b. La variable b corresponde a min y la variable a corresponde a max+1-min, esto debido a que el b se le suma a toda la ecuación entonces debido a esto se le tiene que restar en a. Además que siempre hay que aumentarlo en 1 para que nos de el rango maximo de max. Lo que retorna esta función seria el double numaleatorio.

randp2D: Como se ve en la linea 17, esta función es de tipo p2D y no recibe parámetros. De manera similar a randNum se crean los números aleatorios. La diferencia es que ya se define un rango de min=-50 y max=50.En la linea 24 se define una variable tipo p2D llamada puntoaleatorio, donde después se llena con los 2 numeros aleatorios creados. Se retorna esta variable puntoaleatorio.

dist: Como se ve en la linea 30, esta función es de tipo double y recibe 2 variables tipo p2D.Para obtener esta distancia se utilizó la ecuación 1. En la linea 31 se declar una variable tipo double llamada xtotal, que consiste en la resta de las 2 equis.Para esto primero hay que obtener el valor de cada equix, por ende se agarran las variables de entrada tipo p2D que se llama p1 y p2 y se hace p1.x y p2.x para obtener los valores de las equis en estos puntos. Entonces xtotal nada más resta estos valores, y ytotal realiza lo mismo pero usando p1.y y p2.y. En la linea 33 se define la variable tipo double distancia que correponde a la ecuación 1. Para realizar la raíz cuadrada se usa la función sqrt() y para elevar al cuadrado se usa pow().Finalmente se retorna distancia, la cual corresponde a la distancia euclidiana entre los 2 puntos p1 y p2.

$$distancia = \sqrt{(X_1 - X_2)^2 + (Y_1 - Y_2)^2} \tag{1}$$

randP2DTri: Como se ve en la linea 36, esta es una función tipo p2D, con una entrada tipo p2D llamada p1.Primeramente se declaran 2 variables, uno tipo p2D con etiqueta puntoaleatorio y la otra tipo double con etiqueta distancia con un valor de 101(numero que

posee la condición de ser mayor a 0). En la linea 39 se realiza un while con condición de que siga corriendo si distancia es mayor a diez, lo que significa que cuando la distancia sea menor a 10 el ciclo parará y ya se obtendrá el punto aleatorio que cumple la condición de la distancia. En la linea 41 se le asigna a puntoaleatorio un punto aleatorio que retorna la función previamente creada llamada randP2D(). En la linea 42 se introduce el puntoaleatorio con p1 en la función previamente creada dist que lo que retorna se guardará en la variable distancia. Por ende el ciclo se repite hasta encontrar un puntoaleatorio que cumpla la condición. Por último se retorna este puntoaleatorio.

triIneq: Como se observa en la linea 46 la función es de tipo int, y recibe 3 entradas tipo p2D con etiqueta p1,p2 y p3. La desigualdad triangular tiene que ver con los lados del triangulo, por ende se saca cada lado del triangulo obteniendo la distancia con la función dist de p1,p2 y p3. Estos lados como se pueden ver en de la linea 47 a la 49 son de tipo double. El if que se aplica presenta la condición cuando se cumple la desigualdad triangular por ende en caso de que entre al if se retorna un 1, en caso contrario mediante el else se retorna un 0.

randTri: Como se puede observar en la linea 57, esta función es de tipo tri y posee una entrada tipo p2D con etiqueta p1.Primeramente se definen 3 variables, las primeras tipo p2D con etiqueta p2 y p3 donde además se llenan sus valores .x y .y con 0, y la otra variable que se define es tipo int con etiqueta triIneqq y contenido igual a 0.El primer while que se muestra en la linea 65 posee la condición de parar el ciclo cuando triIneqq sea diferente a 0, ya que en la linea 71 se ve que la variable esta igualada al return de la función triIneq que da 1 en caso que se cumpla la desigualdad, por ende este ciclo se detiene cuando se sabe que el triangulo cumple con la condición de desigualdad. En el while de la linea 67 se cambian los valores de p2 y p3 mediante la función randP2DTri que asegura que el punto nuevo va a tener una distancia menor a 10 del p1, además que se asegura con la condicióndel while que no se presenta el mismo punto por parte de p2 y p3, se nota que el while esta adentro del anterior por ende en caso que no se cumpla la desigualdad triangular se crearían nuevos valores para p2 y p3. Finalmente cuando ya se obtienen los valores de p2 y p3 se declara una variable tipo trialeatorio donde se llena el contenido con .A=p1, .B=p2 y .C=p3. Por ende esta función retorna trialeatorio que corresponde a un triangulo con distancia menor a 10 entre cada punto y que cumple la desigualdad triangular.

calcArea: Como se ve en la linea 81, esta función es de tipo double y recibe una variable tipo tri con etiqueta t1.Primeramente se obtienen los lados del triangulo con la función dist como se ve en las lineas 82, 83 y 84.Para obtener el area se hace mediante la fórmula de Herón como se ve en la ecuación 3, donde s se ve en la ecuación 2 y corresponde al semiperimetro. En la linea 85 se puede ver donde se define un double llamado se que corresponde a la ecuación 2 y otro double llamado area que corresponde a la ecuación 3. cabe mencionar que para la raíz cuadrada se utilizó la función sqrt().Por último se retorna el area.

$$s = \frac{lado1 + lado2 + lado3}{2} \tag{2}$$

$$area = \sqrt{s(s - lado1)(s - lado2)(s - lado3)} \tag{3}$$

reserveTri: Como se ve en la línea 89, esta es una función tipo tri* y recibe un int

con etiqueta cantidadtri que corresponde a la cantidad de variables tipo tri que se quieren almacenar en la memoria heap. Para ello en la linea 90 define la variable tipo tri* que correspondería a la dirección de memoria aparatada.Para ello se usa la función malloc y dentro de esta se utiliza la función sizeof(tri) para que indique el tamaño de un tri y se multiplica por cantidadtri para que malloc reserve el espacio adecuado para que se almacene la cantidad solicitada de variables tipo tri. Al final la función devuelve rutamalloc que corresponde a la dirección de la memoria reservada para la cantidad de tri deseada.

initTri: Como se ve en la linea 93, esta función es de tipo void y recibe un tri* correspondiente a la dirección de la memoria almacenada con etiqueta rutamalloc, y un int que corresponde a la cantidad de variables tipo tris que se quieren y con etiqueta cantidadTri. Primeramente se ve el for que se tiene hasta que i sea igual a cantidadTri.En este for primero se define un p2D con etiqueta de puntoaleatorio que se genera con ayuda de la función randP2D(), posteriormente se crea un tri aleatorio utilizando la función randtri() y recibe puntoaleatorio. Este tri aleatorio se almacena dentro de la dirección de memoria como se observa en la linea 98.

sortTri: Como se ve en la linea 101, esta función es de tipo void, recibe la etiqueta rutamalloc tipo tri* que se sabe que es la dirección de memoria reservada, además del int cantidadtri que corresponde a la cantidad de variables tipo tri que se quieren. Primeramente se definen 3 tri con etiqueta a ,b y c. Posteriormente tres int con etiqueta z,k y t y todas con contenido 0. Además que también se crea un array con la etiqueta de copiarutamalloc del tamaño de cantidadtri.El for de la linea 109 corresponde al que llena el array copiarutamalloc con el contenido de rutamalloc. Por ende el siguiente for de la linea 113 trata de hacer una comparación entre a y b, ya que a se va comparando con todos los elementos del array, el b va cambiando gracias al for de la linea 116, y el if de la linea 119 lo que hace es en caso de que el area sea mayor aumenta z. Cabe mencionar que para obtener el area se utiliza la función calcArea. Por ende el area menor va a poseer un valor de z=0 y su posición correspondería a malloc[0], para el segundo menor malloc[1] y así hasta llegar a cantidadtri-1. En caso de que se tuviera la misma area es que se realiza el if de la linea 125 en dado caso k aumenta 1 cada vez que entra y el if de la linea 128 indica que en caso de que k sea mayor que uno lo que significa que ya es el siguiente elemento con el area repetida, se aumente 1 en t y c sea igual a B. El if de la linea 138 corresponde al caso donde se repitan areas pero si llega un valor menor al de la área repetida este estaría en la posición z correspondiente. En cambio en caso contratrio seria en posición z+t ya que t representa las veces que se repiten las áreas y habría que mover una casilla todos los valores de z.

printTri: Como se muestra en la linea 156 esta función es de tipo void y recibe un tri con etiqueta t1. Para obtener Las coordenadas x y y en el punto A del triangulo p1 simplemente se hace p1.A.x y p1.A.y se imprimen %,1f debido a que es tipo double con un solo decimal. Lo mismo para B y C. Para obtener el area se usa la función calcArea.

printAllTri: Como se muestra en la linea 132, esta función es tipo void y recibe un tri* rutamalloc y un int cantidadtri. El for de la linea 133 va a ir moviendo la memoria de espacio a espacio, entonces eso le llega de parámetro a printTri y el resultado seria la impresión de cada uno de los triángulos.

3. Main.c

```
1  #include <stdio.h>
2  #include "triangulos.h"
3  #include <math.h>
4  #include<stdlib.h>
5  #include <time.h>
6
7  int main(int argc, char const *argv[])
8  {
9      srand(time(NULL));
10     int cantidadtri= atoi(argv[1]);
11     tri* bloq=reserveTri(cantidadtri);
12     initTri(bloq, cantidadtri);
13     sortTri(bloq,cantidadtri);
14     printAllTri(bloq,cantidadtri);
15     free(bloq);
16
17     return 0;
18 }
```

El main resulta muy simple debido a que solo llamamos a las funciones de triangulos.c. Se puede observar como el main también posee todas las bibliotecas que se necesitan además de que se incluye el encabezado de triangulos.h para hacer uso de los tipos de datos creados y todas sus funciones.En la linea 9 se llama srand(time(NULL)) para tener siempre numeros aleatorios distintos.En la linea 10 se extrae de la linea de comandos mediante atoi y se iguala a cantidadtri.En la linea 11 se crea una variable tipo tri* con etiqueta bloq que contiene la dirección de memoria que se va a reservar, esto utilizando la funcion reserveTri con el parámetro de cantidadtri.En la linea 12 se llama a la función initTri enviando las variables bloq y cantidadtri como parámetro para llenar los espacios de memoria reservada con constantes tipo tri.En la linea 13 se mandan los mismos parámetros para ordenar las constantes tipo tri en orden ascendente de acuerdo a su área. En la linea 14 se mandan los mismos parámetros a la función printAllTri para que se imprima la información de todos los triangulos almacenados en la memoria dinámica en orden. Por último en la linea 15 se libera la memoria reservada mediante free(bloq).

4. Makefile

```
1  all:
2          gcc -o triangulos.x main.c triangulos.c -lm
3  clean:
4          rm -f *.o *.x
5  run:
6          ./triangulos.x 10
```

Para el makefile se hicieron los targets all, clean y run. De manera que all compila los binarios y crea el ejecutable triangulos.x el cual ejecutará 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.

En la figura 1,2,3 se muestra el programa corriéndolo mediante el uso de makefile.

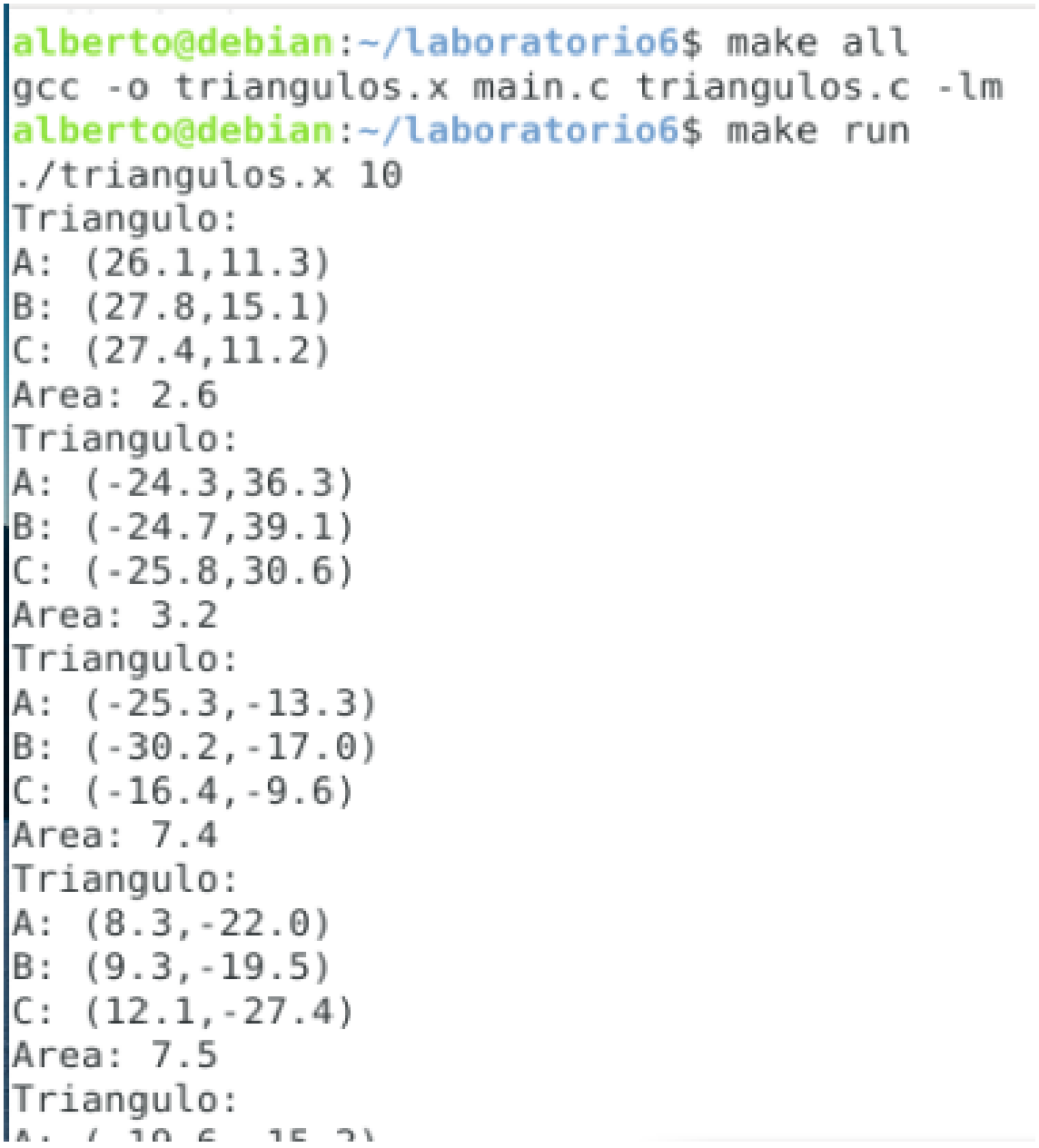

Figura 1: Programa con automatización de compilación

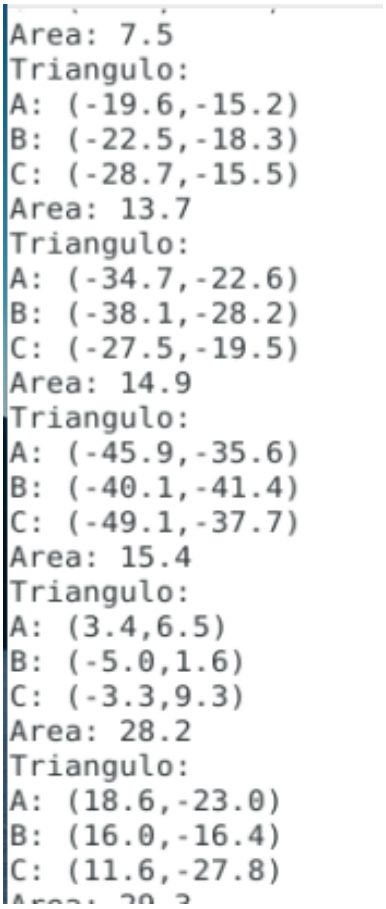

Figura 2: Programa con automatización de compilación

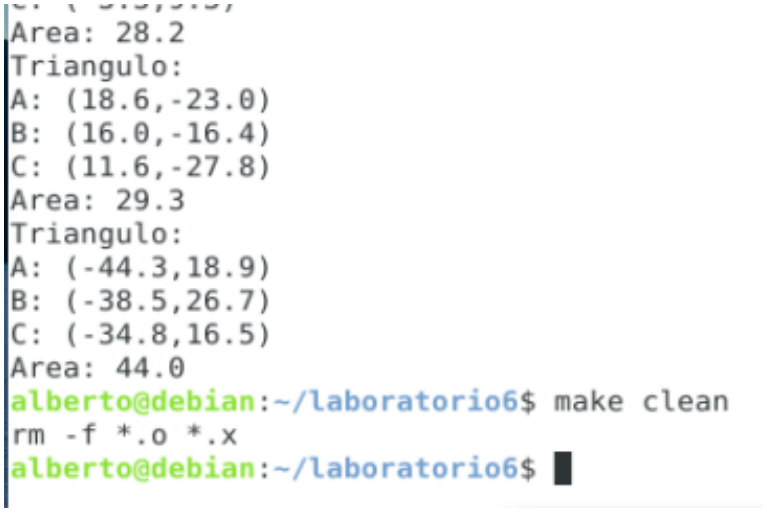

Figura 3: Programa con automatización de compilación