

UNIVERSIDAD DE GUANAJUATO

DIVISIÓN DE INGENIERÍAS CAMPUS IRAPUATO-SALAMANCA

LIC. EN ING. EN SISTEMAS COMPUTACIONALES

ALGORITMOS Y ESTRUCTURAS DE DATOS

PROFESOR: DR. CARLOS HUGO GARCÍA CAPULÍN

NO. DE TAREA: 05

NOMBRE DE LA TAREA:

ESTRUCTURAS (NÚMERO COMPLEJO)

ESTUDIANTE:

MANRÍQUEZ COBIÁN ROGELIO

FECHA DE ENTREGA:

15 DE SEPTIEMBRE DEL 2020



Problema

Implementar un programa que realice las operaciones aritméticas básicas (+, -, *, /) de dos números complejos, utilizando estructuras de datos y funciones.

Para recordar qué es una estructura de datos, se deja la siguiente explicación:

ESTRUCTURAS DE DATOS

Una estructura es una agrupación de datos de diferente tipo que tienen un nombre común.

SINTAXIS para la declaración de la estructura:

```
typedef struct {  
    tipo_de_dato Nombre1;  
    tipo_de_dato Nombre2;  
    ...  
    tipo_de_dato NombreN;  
  
}Nombre_De_La_Estrucura;  
  
Nombre_De_La_Estrucura variable1;
```

Solución Implementada:

```

Clase11.c x Clase12.c x
25 #include <stdio.h>
26 #include <math.h>
27
28 //Declaración de la estructura
29
30 typedef struct {
31     float real;
32     float imga;
33 } num_Complejo;
34
35
36 //Esta función regresa una estructura referente a num_Complejo
37 num_Complejo Captura_numComplejo();
38
39 void muestra_numComplejo(num_Complejo *w);
40
41 num_Complejo sumaComplejo (num_Complejo *w, num_Complejo *z);
42 num_Complejo restaComplejo (num_Complejo *w, num_Complejo *z);
43 num_Complejo producComplejo (num_Complejo *w, num_Complejo *z);
44 num_Complejo diviComplejo (num_Complejo *w, num_Complejo *z);
45
46 int main (void){
47
48     num_Complejo z1, z2, r;
49
50     //Pedimos Ingresar los datos
51     printf ("\nIngrese el numero z1: ");
52     z1 = Captura_numComplejo();
53
54     printf ("\nIngrese el numero z2: ");
55     z2 = Captura_numComplejo();
56
57     //Mostramos los datos capturados
58     printf ("\nLos numeros capturado son: ");
59     muestra_numComplejo(&z1);
60     printf ("\n");
61     muestra_numComplejo(&z2);
62
63     //Mostrar Operaciones Aritmeticas
64     r = sumaComplejo (&z1,&z2);
65     printf ("\n\nLa suma es:");
66     muestra_numComplejo(&r);
67
68     r = restaComplejo(&z1,&z2);
69     printf("\n\nLa resta es: ");
70     muestra_numComplejo(&r);
71
72     r = producComplejo(&z1,&z2);
73     printf("\n\nEl producto es: ");
74     muestra_numComplejo(&r);
75
76     r = diviComplejo(&z1,&z2);
77     printf("\n\nLa division es: ");
78     muestra_numComplejo(&r);
79
80     getchar ();
81     return 0;
82 }
83
84 //Función para la División de Numero Complejo
85 num_Complejo diviComplejo (num_Complejo *w, num_Complejo *z){
86     num_Complejo divi;
87
88     divi.real= ((w->real * z->real) + (w->imga * z->imga)) / (pow(z->real, 2) + pow(z->imga, 2));
89     divi.imga= ((w->imga * z->real) - (w->real * z->imga)) / (pow(z->real, 2) + pow(z->imga, 2));
90
91     return divi;
92 }
93
94 //Función para la Multiplicación de Numero Complejo
95 num_Complejo producComplejo (num_Complejo *w, num_Complejo *z){
96     num_Complejo produc;
97
98     produc.real = (w->real * z->real) - (w->imga * z->imga);
99     produc.imga = (w->real * z->imga) + (w->imga * z->real);
100

```

```

101     return produc;
102 }
103
104 //Función para la Suma de Numero Complejo
105 num_Complejo sumaComplejo (num_Complejo *w, num_Complejo *z){
106     num_Complejo suma;
107
108     suma.real = w->real + z->real;
109     suma.imga = w->imga + z->imga;
110
111     return (suma);
112 }
113
114 //Función para la Diferencia de Numero Complejo
115 num_Complejo restaComplejo (num_Complejo *w, num_Complejo *z){
116     num_Complejo resta;
117
118     resta.real = w->real - z->real;
119     resta.imga = w->imga - z->imga;
120
121     return (resta);
122 }
123
124 //Mostrar los datos capturados
125 void muestra_numComplejo(num_Complejo *w){
126     if (w->imga < 0){
127         printf ("\n%2.1f %2.1f i", w->real, w->imga);
128     }else{
129         printf ("\n%2.1f + %2.1f i", w->real, w->imga);
130     }
131 }
132
133 //Capturar los datos ingresados
134 num_Complejo Captura_numComplejo(){
135     num_Complejo w;
136
137     printf ("\nReal: ");
138     scanf ("%f", &w.real); fflush (stdin);
139
140     printf ("Imaginaria: ");
141     scanf ("%f", &w.imga); fflush (stdin);
142
143     return w;
144 }
145
146

```

Pruebas y Resultados:

Ahora que ya tenemos el problema planteado hay que implementar una solución de acuerdo con lo que se nos pide resolver en el reporte.

Iniciamos creando un nuevo archivo en nuestro editor de textos Notepad ++ el cual lo guardaremos con el nombre de T05.c ("En mi caso tiene el nombre de Clase11, ya que este ejercicio se planteó en la clase).

Comenzamos escribiendo nuestro **"#include <stdio.h>"** y **"#include <math.h>"** esta librería "math" sirve principalmente para operaciones matemáticas más avanzadas como raíz cuadrada, elevar un número a la "n" potencia, etc.

Ahora, lo interesante será que después de escribir nuestras librerías de preprocesador tendremos que crear nuestra estructura de datos, el cuál está declarado con la nomenclatura **"typedef struct"** para evitarnos una cosa tediosa en el código **"main"**. Dentro de nuestra estructura declararemos dos variables de tipo **"float"**, el primero tendrá como nombre **"real"** y el otro tendrá como nombre **"imga"**, por último, declaramos nuestra estructura con el nombre de **"numero_Complejo"**.

Entonces, escribiremos nuestra función **"main"**, donde declararemos nuestra estructura de datos con el nombre de tres variables que estaremos usando para las funciones y operaciones dentro de estas operaciones de números complejos, estas variables serán **z1, z2, r**.

Para la primer parte de nuestro código haremos el usuario ingrese los números para guardarlos en la variable **"z1 y z2"** en el cual se mandará a una función **"Captura_numComplejo"** que esta función misma se declarará dentro una variable **"num_Complejo"** en **"w"**, donde guardemos los números reales e imaginarios dentro de la variable **"w"** que está apuntando a la estructura **"num_Complejo"** y por último retornamos la dirección de **"w"**.

NOTA: Declarar la función **num_Complejo Captura_numComplejo ()** como prototipo.

Ahora, lo que hacemos es mostrar los números capturados por la función, en otra función vacía llamada **"void muestra_numComplejo (num_Complejo *w)"** en la cuál tenemos almacenado los valores gracias al apuntador **"w"**; ahora solo dentro de esta función tendremos una condición para la poder imprimir en pantalla los datos capturados, la condición es de que el apuntador **"w"** dirigido a la variable **"imga"** de la estructura es menor a cero, imprima la parte positiva (real) y la parte negativa (imaginaria), sino, imprime ambos números complejos de manera positiva.

Esta función solo devolverá los valores a la función **"main"** para mostrar la dirección en memoria que almacena **"z1 y z2"**.

NOTA: Declarar la función **void muestra_numComplejo (num_Complejo *w)** como prototipo.

Entonces, llega lo más importante del problema, implementar una solución en las operaciones aritméticas.

Por primera instancia, resolveremos, la operación “**suma**”.

Esta función tendrá como valor un “**num_Complejo sumaComplejo (num_Complejo *w, num_Complejo *z)**” dentro como argumento de nuestra función recibirá dos parámetros a manera de apuntadores para poder almacenar los datos que realicemos con ellos de manera dinámica.

Dentro de nuestra función declaremos una variable de tipo “**num_Complejo**” con el nombre “**suma**”, ahora si recordamos nuestras clases de aritmética de la secundaria, para hacer la suma de dos números complejos hay una regla especial, ya que no se podrá hacer la suma típica que conocemos. Esta suma esta dividida en dos partes las cuales se tendrán que sumar los números reales solo con números reales y la suma de números imaginarios.

Entonces, hacemos la operación para “**suma.real**” en la cual tendrá como operación aritmética la suma de solo los números reales de “**w, z**” apuntando de manera dinámica a la estructura “**num_Complejo**” y de igual manera para “**suma.imga**” en los valores de “**w, z**” que es la parte imaginaria.

Ejemplo:

```
suma.real = w->real + z->real;
```

```
suma.imga = w->imga + z->imga;
```

Por último, retornamos el valor de “**suma**”.

NOTA: Declarar la función **num_Complejo sumaComplejo (num_Complejo *w, num_Complejo *z)** como prototipo.

Ahora, utilizaremos nuestra variable “**r**” haciendo referencia a “**resultado**” de las operaciones que vayamos a realizar e irlas guardando en la variable, entonces, “**r**” será a igual a la función “**sumaComplejo (&z1, &z2)**” teniendo como argumentos las direcciones de memoria de dichos números, ahora imprimiremos en pantalla el valor de dirección de memoria que tiene ahora “**r**” con la ayuda de la función “**muestra_numComplejo (&r)**”.

Haremos el mismo procedimiento para la operación “**resta**”.

Esta función tendrá como valor un “**num_Complejo restaComplejo (num_Complejo *w, num_Complejo *z)**” dentro como argumento de nuestra función recibirá dos parámetros a manera de apuntadores para poder almacenar los datos que realicemos con ellos de manera dinámica.

Dentro de nuestra función declaremos una variable de tipo “**num_Complejo**” con el nombre “**resta**”.

Entonces, hacemos la operación para “**resta.real**” en la cual tendrá como operación aritmética la resta de solo los números reales de “**w, z**” apuntando de manera dinámica a la estructura “**num_Complejo**” y de igual manera para “**resta.imga**” en los valores de “**w, z**” que es la parte imaginaria.

Ejemplo:

```
resta.real = w->real - z->real;
resta.imga = w->imga - z->imga;
```

Por último, retornamos el valor de “**resta**”.

NOTA: Declarar la función **num_Complejo restaComplejo (num_Complejo *w, num_Complejo *z)** como prototipo.

Ahora, utilizaremos nuestra variable “**r**” haciendo referencia a “**resultado**” de las operaciones que vayamos a realizar e ir las guardando en la variable, entonces, “**r**” será a igual a la función “**restaComplejo (&z1, &z2)**” teniendo como argumentos las direcciones de memoria de dichos números, ahora imprimiremos en pantalla el valor de dirección de memoria que tiene ahora “**r**” con la ayuda de la función “**muestra_numComplejo (&r)**”.

Haremos el mismo procedimiento para la operación “**producto**”.

Esta función tendrá como valor un “**num_Complejo** **producComplejo (num_Complejo *w, num_Complejo *z)**” dentro como argumento de nuestra función recibirá dos parámetros a manera de apuntadores para poder almacenar los datos que realicemos con ellos de manera dinámica.

Dentro de nuestra función declaremos una variable de tipo “**num_Complejo**” con el nombre “**produc**”.

Entonces, hacemos la operación para “**produc.real**” en la cual tendrá como operación aritmética el producto de solo los números reales de “**w, z**” menos el producto de los números imaginarios de “**w, z**” apuntando de manera dinámica a la estructura “**num_Complejo**” y de igual manera para “**produc.imga**” se hará la operación aritmética del producto real de “**w**” por el imaginario de “**z**” menos el producto del valor de “**w**” imaginaria por el valor real de “**z**”.

Ejemplo:

```
produc.real = (w->real * z->real) - (w->imga * z->imga);
produc.imga = (w->real * z->imga) + (w->imga * z->real);
```

Por último, retornamos el valor de “**produc**”.

NOTA: Declarar la función **num_Complejo** **producComplejo (num_Complejo *w, num_Complejo *z)** como prototipo.

Ahora, utilizaremos nuestra variable “**r**” haciendo referencia a “**resultado**” de las operaciones que vayamos a realizar e ir las guardando en la variable, entonces, “**r**” será a igual a la función “**producComplejo (&z1, &z2)**” teniendo como argumentos las direcciones de memoria de dichos números, ahora imprimiremos en pantalla el valor de dirección de memoria que tiene ahora “**r**” con la ayuda de la función “**muestra_numComplejo (&r)**”.

Haremos el mismo procedimiento para la operación “**división**”.

Esta función tendrá como valor un “**num_Complejo diviComplejo (num_Complejo *w, num_Complejo *z)**” dentro como argumento de nuestra función recibirá dos parámetros a manera de apuntadores para poder almacenar los datos que realicemos con ellos de manera dinámica.

Dentro de nuestra función declaremos una variable de tipo “**num_Complejo**” con el nombre “**divi**”.

Entonces, hacemos la operación para “**divi.real**” en la cual tendrá como operación aritmética el producto de solo los números reales de “**w, z**” más el producto de los números imaginarios “**w, z**” entre el número real “**z**” al cuadrado, más el número imaginario “**z**” al cuadrado, apuntando de manera dinámica a la estructura “**num_Complejo**” y de igual manera para “**divi.imga**” se hará la operación aritmética del producto imaginario de “**w**” por el real de “**z**” menos el producto del valor real de “**w**” imaginaria por el valor imaginario de “**z**” entre el número real de “**z**” al cuadrado, más el número imaginario de “**z**” al cuadrado.

Ejemplo:

```
divi.real= ((w->real * z->real) + (w->imga * z->imga)) / (pow(z->real, 2) + pow(z->imga, 2));
```

```
divi.imga= ((w->imga * z->real) - (w->real * z->imga)) / (pow(z->real, 2) + pow(z->imga, 2));
```

Por último, retornamos el valor de “**divi**”.

NOTA: Declarar la función **num_Complejo diviComplejo (num_Complejo *w, num_Complejo *z)** como prototipo.

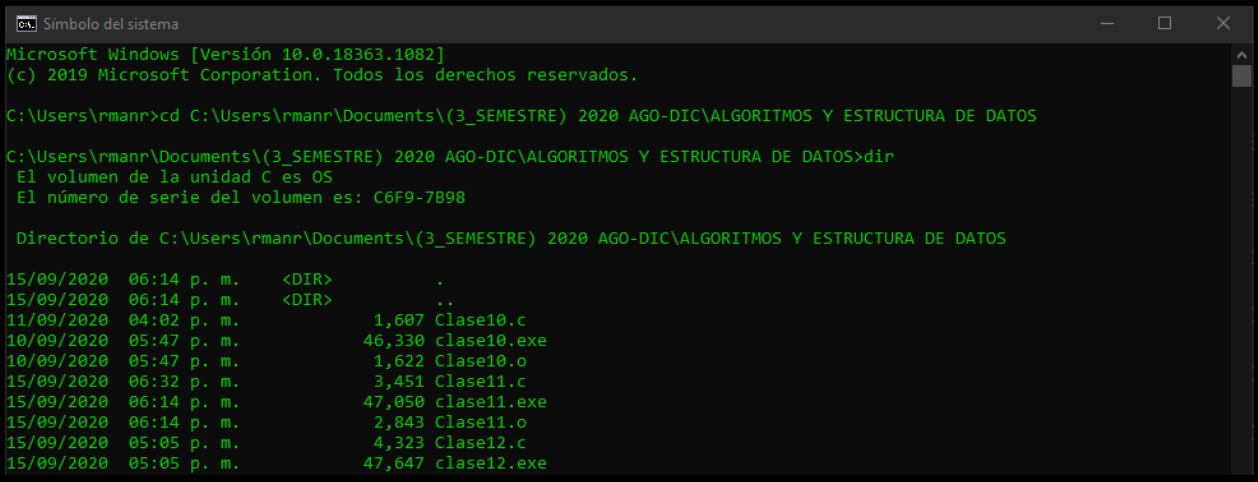
Ahora, utilizaremos nuestra variable “**r**” haciendo referencia a “**resultado**” de las operaciones que vayamos a realizar e irlas guardando en la variable, entonces, “**r**” será a igual a la función “**diviComplejo (&z1, &z2)**” teniendo como argumentos las direcciones de memoria de dichos números, ahora imprimiremos en pantalla el valor de dirección de memoria que tiene ahora “**r**” con la ayuda de la función “**muestra_numComplejo (&r)**”.

Con esto finalizamos nuestro bloque de código, y ahora nos toca probarlo para observar su funcionamiento.

Para comprobar que nuestro programa realiza lo pedido, abriremos nuestro “**CMD**” he ingresaremos hasta la carpeta donde se tiene guardado el archivo; en mi caso se encuentra en la dirección:

- *C:\Users\rmanr\Documents\{(3_SEMESTRE) 2020 AGO-DIC\ALGORITMOS Y ESTRUCTURA DE DATOS*

Y con el comando **<dir>** veremos que el archivo se ha guardado de manera satisfactoria.



```

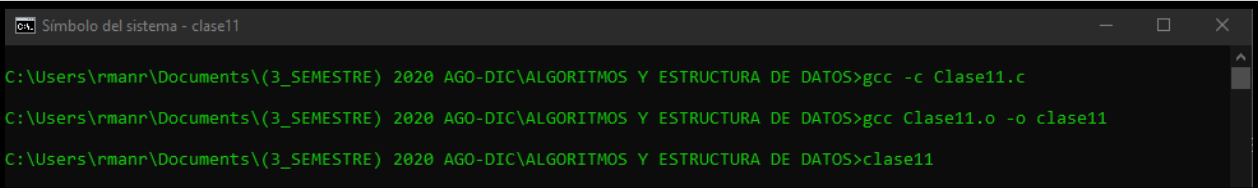
C:\Users\rmanr>cd C:\Users\rmanr\Documents\{(3_SEMESTRE) 2020 AGO-DIC\ALGORITMOS Y ESTRUCTURA DE DATOS

C:\Users\rmanr\Documents\{(3_SEMESTRE) 2020 AGO-DIC\ALGORITMOS Y ESTRUCTURA DE DATOS>dir
El volumen de la unidad C es OS
El número de serie del volumen es: C6F9-7B98

Directorio de C:\Users\rmanr\Documents\{(3_SEMESTRE) 2020 AGO-DIC\ALGORITMOS Y ESTRUCTURA DE DATOS

15/09/2020  06:14 p. m.      <DIR>          .
15/09/2020  06:14 p. m.      <DIR>          ..
11/09/2020  04:02 p. m.             1,607 Clase10.c
10/09/2020  05:47 p. m.          46,330 clase10.exe
10/09/2020  05:47 p. m.             1,622 Clase10.o
15/09/2020  06:32 p. m.             3,451 Clase11.c
15/09/2020  06:14 p. m.          47,050 clase11.exe
15/09/2020  06:14 p. m.             2,843 Clase11.o
15/09/2020  05:05 p. m.             4,323 Clase12.c
15/09/2020  05:05 p. m.          47,647 clase12.exe
  
```

Ahora tendremos que compilar nuestro código con los siguientes comandos que se muestran en la imagen:



```

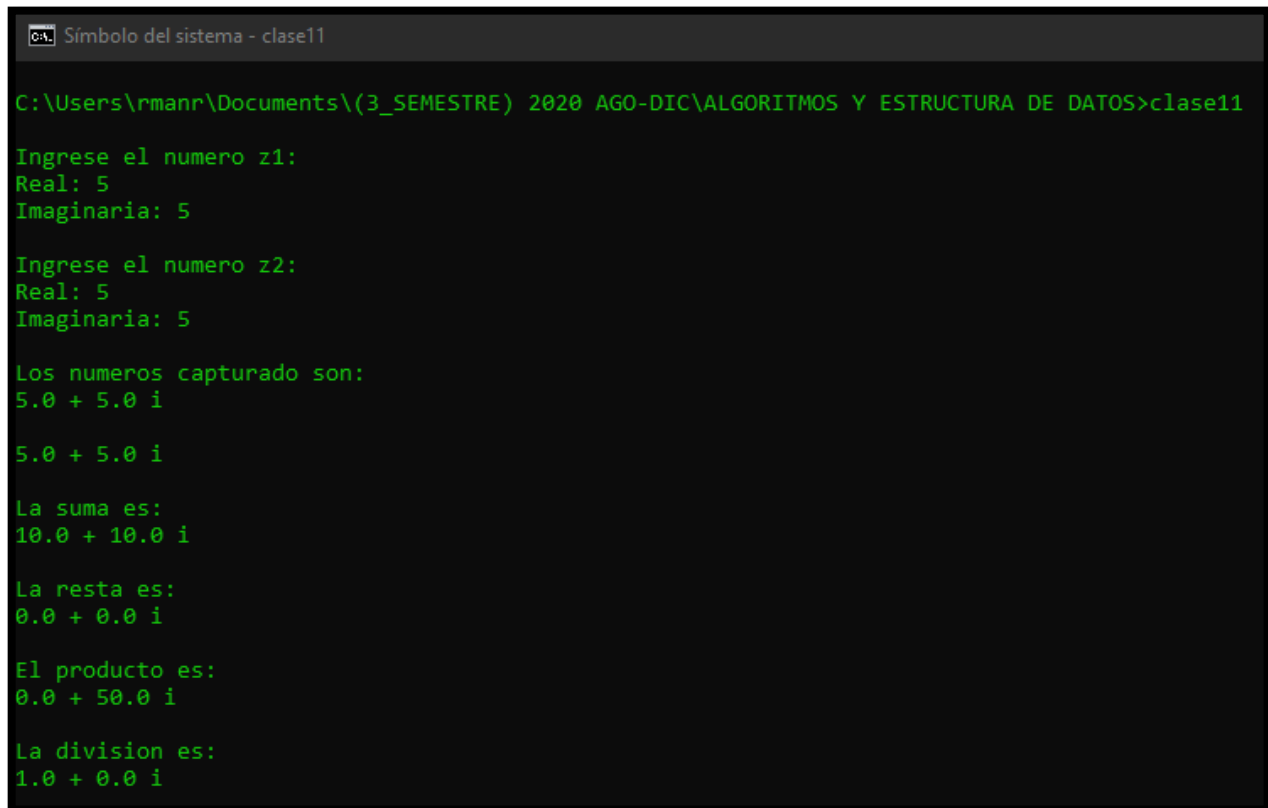
C:\Users\rmanr\Documents\{(3_SEMESTRE) 2020 AGO-DIC\ALGORITMOS Y ESTRUCTURA DE DATOS>gcc -c Clase11.c

C:\Users\rmanr\Documents\{(3_SEMESTRE) 2020 AGO-DIC\ALGORITMOS Y ESTRUCTURA DE DATOS>gcc Clase11.o -o clase11

C:\Users\rmanr\Documents\{(3_SEMESTRE) 2020 AGO-DIC\ALGORITMOS Y ESTRUCTURA DE DATOS>clase11
  
```

Si todo salió bien, solamente dará líneas de salto significando que todo el proceso de compilación y enlazamiento salió bien; de lo contrario, escribiste algún comando mal o tu código tiene algún error de sintaxis.

Ahora ejecutaremos nuestro programa para realizar las pruebas y observar si es lo que se quiso resolver desde un principio:



```

C:\Users\rmanr\Documents\3_SEMESTRE) 2020 AGO-DIC\ALGORITMOS Y ESTRUCTURA DE DATOS>clase11

Ingrese el numero z1:
Real: 5
Imaginaria: 5

Ingrese el numero z2:
Real: 5
Imaginaria: 5

Los numeros capturado son:
5.0 + 5.0 i

5.0 + 5.0 i

La suma es:
10.0 + 10.0 i

La resta es:
0.0 + 0.0 i

El producto es:
0.0 + 50.0 i

La division es:
1.0 + 0.0 i
  
```

El programa nos pide ingresar para la variable “**z1**” los valores **reales** e **imaginarios**.

Ingresaremos el valor “5” para **real** e **imaginario**.

De igual manera ingresaremos en la variable “**z2**” los valores **reales** e **imaginarios**.

Ingresaremos el valor “5” para **real** e **imaginario**.

Después se imprimirá en pantalla los valores que hemos ingresado en las dos variables, en las secciones reales e imaginarias.

Luego, se imprimen los valores reales e imaginarias de las operaciones aritméticas que hemos desarrollado en código, para tener una vista mejor de lo que está pasando dentro de cada operación aritmética se explicará en seguida de manera breve.

Suma:

suma.real = w->real + z->real; ----- suma.real = 5 + 5 ----- suma.real = 10

suma.imga = w->imga + z->imga; ----- suma.imga = 5 + 5 ----- suma.imga = 10

La suma es: 10 + 10 i

Resta:

resta.real = w->real - z->real; ----- resta.real = 5 - 5 ----- resta.real = 0

resta.imga = w->imga - z->imga; ----- resta.imga = 5 - 5 ----- resta.imga = 0

La resta es: 0 + 0 i

Producto:

produc.real = (w->real * z->real) - (w->imga * z->imga);

produc.real = (5 * 5) - (5 * 5)

produc.real = 25 - 25

produc.real = 0

produc.imga = (w->real * z->imga) + (w->imga * z->real);

produc.imga = (5 * 5) + (5 * 5)

produc.imga = 25 + 25

produc.imga = 50

El producto es: 0 + 50 i

División:

divi.real= ((w->real * z->real) + (w->imga * z->imga)) / (pow(z->real, 2) + pow(z->imga, 2));

divi.real = (5 * 5) + (5 * 5) / (5^2) + (5^2)

divi.real = 50 / 50

divi.real = 1

divi.imga= ((w->imga * z->real) - (w->real * z->imga)) / (pow(z->real, 2) + pow(z->imga, 2));

divi.imga = (5 * 5) - (5 * 5) / (5^2) + (5^2)

divi.imga = 0 / 50

divi.imga = 0

La división es= 1 + 0 i

Con esto queda resuelto nuestro problema que se planteó desde un principio.

Como conclusión se puede mencionar que es muy fácil de trabajar con apuntadores y más en este ejercicio ya que siempre utilizamos el paso de valor mediante referencia para que no se pierda esta práctica que nos ayudará en los siguientes ejercicios que veamos en las clases y para otros proyectos que nosotros podamos llevar a cabo.

Este problema me ha sido de mucho agrado ya que hemos utilizado la parte de programación y la parte matemática para llegar a resolver un tema de números complejos, esperemos seguir con este ritmo para aprender más cosas acerca de apuntadores y memoria dinámica.