Frank Dong
250847142

a)

```
struct complex_tag
{
      double real;
      double imaginary;
};

typedef struct
{
      double real;
      double imaginary;
} Complex_type;
```

b)

```
    Complex_type multiply(struct complex_tag c1, struct 
complex_tag c2)
    {
        //Declaration of return variable
        Complex_type value;

        //Multiplication Calculation
        value.real = c1.real * c2.real - c1.imaginary * 
c2.imaginary;
        value.imaginary = c2.real * c1.imaginary + c1.real * 
c2.imaginary;

        return value;
    }
```

d)

```
int divide(struct complex_tag *c1, struct complex_tag *c2, struct
complex_tag *result)
{
      //If a2^2 + b2^2 = 0 return error
      if (c2->real * c2->real + c2->imaginary * c2->imaginary == 0)
      {
            return -2;
      }

      //Else division calculation and return 0
      result->real = (c1->real * c2->real + c1->imaginary * c2-
>imaginary) / (c2->real * c2->real + c2->imaginary * c2->imaginary);
      result->imaginary = (c2->real * c1->imaginary - c1->real * c2-
>imaginary) / (c2->real * c2->real + c2->imaginary * c2->imaginary);
      return 0;
}
```

e)

operation_function.c

```
//Author: Frank Dong
//Purpose: To create a the functions which will calculate the
multiplication, division, addition and difference
//            of a 2 complex numbers.
//Date: Nov 30, 2016

#include <stdio.h>
#include <stdlib.h>
#include "operation_function.h"

/*
Name: Frank Dong
Date: Nov 29, 2016
Purpose: Multiply function which will calculate the multiplication of
2 complex numbers
Input: complex_tag c1 & complex_tag c2
Output: value (Complex_type)
*/
Complex_type multiply(struct complex_tag c1, struct complex_tag c2)
{
      //Declaration of return variable
      Complex_type value;

      //Multiplication Calculation
      value.real = c1.real * c2.real - c1.imaginary * c2.imaginary;
      value.imaginary = c2.real * c1.imaginary + c1.real *
c2.imaginary;

      return value;
}

/*
Name: Frank Dong
Date: Nov 29, 2016
Purpose: Divide function which will calculate the division of 2
complex numbers
Input: complex_tag *c1, complex_tag *c2, complex_tag *result
Output: -2 (if error) or 0 | calculated value is returned through
pointer
*/
int divide(struct complex_tag *c1, struct complex_tag *c2, struct
complex_tag *result)
{
      //If a2^2 + b2^2 = 0 return error
      if (c2->real * c2->real + c2->imaginary * c2->imaginary == 0)
      {
            return -2;
      }
```

```
    //Else division calculation and return 0
    result->real = (c1->real * c2->real + c1->imaginary * c2-
>imaginary) / (c2->real * c2->real + c2->imaginary * c2->imaginary);
    result->imaginary = (c2->real * c1->imaginary - c1->real * c2-
>imaginary) / (c2->real * c2->real + c2->imaginary * c2->imaginary);
    return 0;
}

/*
Name: Frank Dong
Date: Nov 29, 2016
Purpose: add and subtract function which will calculate the addition
and difference of 2 complex numbers
Input: complex_tag c1, complex_tag c2, complex_tag **sum, &
complex_tag **diff
Output: -1 (if error) or 0 | calculated values is returned through
pointers
*/
int add_and_sub(struct complex_tag c1, struct complex_tag c2, struct
complex_tag **sum, struct complex_tag **diff)
{
    //Set *sum and *diff equalled to (typecast) allocation of memory
size of complex_tag
    *sum = (struct complex_tag *)malloc(sizeof(struct complex_tag));
    *diff = (struct complex_tag *)malloc(sizeof(struct complex_tag));

    //If memory cannot be allocated, print error and return -1
    if (sum == 0 || diff == 0)
    {
        printf("Error memory allocation error");
        return -1;
    }

    //Sum and difference calculation
    (*sum)->real = c1.real + c2.real;
    (*sum)->imaginary = c1.imaginary + c2.imaginary;
    (*diff)->real = c1.real - c2.real;
    (*diff)->imaginary = c1.imaginary - c2.imaginary;
    return 0;
}
```

<u>operation_function.h</u>

```
//Author: Frank Dong
//Purpose: To create a program which will calculate the
multiplication, division, addition and difference
//            of a 2 complex numbers.
//Date: Nov 30, 2016

#ifndef OPERATION_FUNCTION
#define OPERATION_FUNCTION
```

```
//Declare complex_tag structure
struct complex_tag
{
	double real;
	double imaginary;
};

//Delcare Complex_type type
typedef struct
{
	double real;
	double imaginary;
} Complex_type;

//Functions
Complex_type multiply(struct complex_tag c1, struct complex_tag c2);
int divide(struct complex_tag *c1, struct complex_tag *c2, struct
complex_tag *result);
int add_and_sub(struct complex_tag c1, struct complex_tag c2, struct
complex_tag **sum, struct complex_tag **diff);

#endif
```

f)

operation.c

```
//Author: Frank Dong
//Purpose: To create a program which will calculate the
multiplication, division, addition and difference
//            of a 2 complex numbers.
//Date: Nov 30, 2016

#include <stdio.h>
#include <stdlib.h>
#include "operation_function.h"

int main(int argc, char *argv[])
{
      //Variable declaration
      struct complex_tag c1;
      struct complex_tag c2;
      Complex_type mutiplyValue;
      struct complex_tag *divideValue = malloc((sizeof(struct
complex_tag)));
      struct complex_tag *sumValue;
      struct complex_tag *diffValue;

      //Checks to see if there are valid amount of arguments
      if (argc != 5)
      {
           printf("Invalid number of arguments! (Exactly 4 arugments
please)");
           exit (-1);
      }

      //Assigns each argument to a variable
      c1.real = atof(argv[1]);
      c1.imaginary = atof(argv[2]);
      c2.real = atof(argv[3]);
      c2.imaginary = atof(argv[4]);

      //Display the numbers user has entered
      printf("The first complex number you have entered is: %f + i(%f)
\n", c1.real, c1.imaginary);
      printf("The second complex number you have entered is: %f + i(%f)
\n", c2.real, c2.imaginary);
      printf("\n");

      //Calls multipy function and displays results
      mutiplyValue = multiply(c1, c2);
      printf("Multiplication: %f + i(%f) \n", mutiplyValue.real,
mutiplyValue.imaginary);
```

```
      //If divide function produces error, return error message. Else 
display results
      if (divide(&c1, &c2, divideValue) == -2)
      {
            printf("Error in division \n");
      }
      else
      {
            divide(&c1, &c2, divideValue);
            printf("Division: %f + i(%f) \n", divideValue->real, 
divideValue->imaginary);
      }

      //Calls the add and subtract function, and displays results
      add_and_sub(c1, c2, &sumValue, &diffValue);
      printf("Addition: %f + i(%f) \n", sumValue->real, sumValue-
>imaginary);
      printf("Difference: %f + i(%f) \n", diffValue->real, diffValue-
>imaginary);
      printf("=========================================================
===== \n");
      printf("\n");

      return 0;
}
```

g)

makefile

```
#format is target-name: target dependencies
#{-tab-}actions

# MACRO definitions
CC  = gcc
CFLAG = -std=c99 -Wall

# All Targets
all: operation

#Executable operation depends on the files operation.o
operation_function.o
operation: operation.o operation_function.o
      $(CC) $(CFLAG) -o operation operation.o operation_function.o

# operation.o depends on the source and header files
operation.o: operation.c operation_function.h
      $(CC) $(CFLAG) -c operation.c

# operation_function.o depends on the source and header files
operation_function.o: operation_function.c operation_function.h
      $(CC) $(CFLAG) -c operation_function.c

# test cases
test: operation
           operation 1 2 3 4
           operation -5 0 -6 0
           operation 0 4 0 6
           operation 5 8 0 -1
           operation 0 5.5 0.25 5
           operation 0 0 5 5
           operation 6 6 0 0

#Clean the build directory
clean:
      rm -f *.o operation
```

h)

*For read me file, please refer to attached file readme.txt

```
> make test
operation 1 2 3 4
The first complex number you have entered is: 1.000000 + i(2.000000)
The second complex number you have entered is: 3.000000 + i(4.000000)

Multiplication: -5.000000 + i(10.000000)
Division: 0.440000 + i(0.080000)
Addition: 4.000000 + i(6.000000)
Difference: -2.000000 + i(-2.000000)
==============================================================

operation -5 0 -6 0
The first complex number you have entered is: -5.000000 + i(0.000000)
The second complex number you have entered is: -6.000000 + i(0.000000)

Multiplication: 30.000000 + i(-0.000000)
Division: 0.833333 + i(0.000000)
Addition: -11.000000 + i(0.000000)
Difference: 1.000000 + i(0.000000)
==============================================================

operation 0 4 0 6
The first complex number you have entered is: 0.000000 + i(4.000000)
The second complex number you have entered is: 0.000000 + i(6.000000)

Multiplication: -24.000000 + i(0.000000)
Division: 0.666667 + i(0.000000)
Addition: 0.000000 + i(10.000000)
Difference: 0.000000 + i(-2.000000)
==============================================================

operation 5 8 0 -1
The first complex number you have entered is: 5.000000 + i(8.000000)
The second complex number you have entered is: 0.000000 + i(-1.000000)

Multiplication: 8.000000 + i(-5.000000)
Division: -8.000000 + i(5.000000)
Addition: 5.000000 + i(7.000000)
Difference: 5.000000 + i(9.000000)
==============================================================

operation 0 5.5 0.25 5
The first complex number you have entered is: 0.000000 + i(5.500000)
The second complex number you have entered is: 0.250000 + i(5.000000)

Multiplication: -27.500000 + i(1.375000)
Division: 1.097257 + i(0.054863)
Addition: 0.250000 + i(10.500000)
```

```
Difference: -0.250000 + i(0.500000)
==============================================================

operation 0 0 5 5
The first complex number you have entered is: 0.000000 + i(0.000000)
The second complex number you have entered is: 5.000000 + i(5.000000)

Multiplication: 0.000000 + i(0.000000)
Division: 0.000000 + i(0.000000)
Addition: 5.000000 + i(5.000000)
Difference: -5.000000 + i(-5.000000)
==============================================================

operation 6 6 0 0
The first complex number you have entered is: 6.000000 + i(6.000000)
The second complex number you have entered is: 0.000000 + i(0.000000)

Multiplication: 0.000000 + i(0.000000)
Error in division
Addition: 6.000000 + i(6.000000)
Difference: 6.000000 + i(6.000000)
==============================================================
```