**Testcase results:**

2 complex numbers:

> operation 2 4 6 8

ENTERED COMPLEX NUMBERS:  
(2.000000) + (4.000000)i  
(6.000000) + (8.000000)i

DIVISION RESULTS:  
(0.440000) + (0.080000)i

MULTIPLICATION RESULTS:  
(-20.000000) + (40.000000)i

ADDITION RESULTS:  
(8.000000) + (12.000000)i

SUBTRACTION RESULTS:  
(-4.000000) + (-4.000000)i

2 real numbers:

> operation 3 0 5 0

ENTERED COMPLEX NUMBERS:  
(3.000000) + (0.000000)i  
(5.000000) + (0.000000)i

DIVISION RESULTS:  
(0.600000) + (0.000000)i

MULTIPLICATION RESULTS:  
(15.000000) + (0.000000)i

ADDITION RESULTS:  
(8.000000) + (0.000000)i

SUBTRACTION RESULTS:  
(-2.000000) + (0.000000)i

2 imaginary numbers:

> operation 0 3 0 5

ENTERED COMPLEX NUMBERS:  
(0.000000) + (3.000000)i  
(0.000000) + (5.000000)i

DIVISION RESULTS:  
(0.600000) + (0.000000)i

MULTIPLICATION RESULTS:  
(-15.000000) + (0.000000)i

ADDITION RESULTS:  
(0.000000) + (8.000000)i

SUBTRACTION RESULTS:  
(0.000000) + (-2.000000)i

A real and an imaginary number:

> operation 4 0 0 4

ENTERED COMPLEX NUMBERS:  
(4.000000) + (0.000000)i  
(0.000000) + (4.000000)i

DIVISION RESULTS:  
(0.000000) + (-1.000000)i

MULTIPLICATION RESULTS:  
(0.000000) + (16.000000)i

ADDITION RESULTS:  
(4.000000) + (4.000000)i

SUBTRACTION RESULTS:  
(4.000000) + (-4.000000)i

An imaginary and a real number:

> operation 0 4 4 0

ENTERED COMPLEX NUMBERS:  
(0.000000) + (4.000000)i  
(4.000000) + (0.000000)i

DIVISION RESULTS:  
(0.000000) + (1.000000)i

MULTIPLICATION RESULTS:  
(0.000000) + (16.000000)i

ADDITION RESULTS:  
(4.000000) + (4.000000)i

SUBTRACTION RESULTS:  
(-4.000000) + (4.000000)i

A zero and a complex number

> operation 0 0 1 2

ENTERED COMPLEX NUMBERS:  
(0.000000) + (0.000000)i  
(1.000000) + (2.000000)i

DIVISION RESULTS:  
(0.000000) + (0.000000)i

MULTIPLICATION RESULTS:  
(0.000000) + (0.000000)i

ADDITION RESULTS:  
(1.000000) + (2.000000)i

SUBTRACTION RESULTS:  
(-1.000000) + (-2.000000)i

A complex number and a zero:

> operation 1 2 0 0

ENTERED COMPLEX NUMBERS:  
(1.000000) + (2.000000)i  
(0.000000) + (0.000000)i

DIVISION RESULTS:  
Can't divide, denomenator = 0

MULTIPLICATION RESULTS:  
(0.000000) + (0.000000)i

ADDITION RESULTS:  
(1.000000) + (2.000000)i

SUBTRACTION RESULTS:  
(1.000000) + (2.000000)i

**C Code - operation\_functions.h**

#ifndef OPERATION\_FUNCTIONS\_H\_

#define OPERATION\_FUNCTIONS\_H\_

//structure for question a)

struct complex\_tag

{

double real;

double imaginary;

};

//structure for question b)

typedef struct

{

double real;

double imaginary;

} Complex\_type;

//prototypes for functions

Complex\_type multiply(struct complex\_tag x, struct complex\_tag y);

int divide(struct complex\_tag \*a, struct complex\_tag \*b, struct complex\_tag \*c);

int add\_subtract(struct complex\_tag a, struct complex\_tag b, struct complex\_tag \*\*c, struct complex\_tag \*\*d);

#endif

**C Code - operation\_functions.c**

#include <stdio.h>

#include <stdlib.h>

#include "operation\_functions.h"

//multiplication function for question b)

Complex\_type multiply(struct complex\_tag x, struct complex\_tag y){

double real = (x.real\*y.real)-(x.imaginary\*y.imaginary);

double imaginary = (y.real\*x.imaginary)+(x.real\*y.imaginary);

Complex\_type c;

c.real = real;

c.imaginary = imaginary;

return c;

}

//division function for question c)

int divide(struct complex\_tag \*a, struct complex\_tag \*b, struct complex\_tag \*c){

double division\_real = (((\*a).real\*(\*b).real)+((\*a).imaginary\*(\*b).imaginary))/(((\*b).real\*(\*b).real)+((\*b).imaginary\*(\*b).imaginary));

double division\_imaginary = (((\*b).real\*(\*a).imaginary)-((\*a).real\*(\*b).imaginary))/(((\*b).real\*(\*b).real)+((\*b).imaginary\*(\*b).imaginary));

(\*c).real = division\_real;

(\*c).imaginary = division\_imaginary;

//if denomenator is 0, then there will be a division error

if ((((\*b).real\*(\*b).real)+((\*b).imaginary\*(\*b).imaginary)) == 0){

return -2;

} else {

return 0;

}

}

//addition and subtraction function for question d)

int add\_subtract(struct complex\_tag a, struct complex\_tag b, struct complex\_tag \*\*c, struct complex\_tag \*\*d){

//dynamic memory allocation

struct complex\_tag \*sum, \*diff;

sum = malloc(sizeof(struct complex\_tag));

diff = malloc(sizeof(struct complex\_tag));

if (!sum || !diff){

printf("Error in allocating memory\n");

return -1;

}

//calculate sum

double sum\_real = a.real + b.real;

double sum\_imaginary = a.imaginary + b.imaginary;

//calculate difference

double diff\_real = a.real - b.real;

double diff\_imaginary = a.imaginary - b.imaginary;

(\*sum).real = sum\_real;

(\*sum).imaginary = sum\_imaginary;

(\*diff).real = diff\_real;

(\*diff).imaginary = diff\_imaginary;

//return created structures to the last 2 parameters

\*c = sum;

\*d = diff;

free(sum);

free(diff);

return 0;

}

**C Code – operation.c**

#include <stdio.h>

#include <stdlib.h>

#include "operation\_functions.h"

int main(int argc, char \*argv[])

{

//variables that start with \_ are pointers

//variables that start and end with \_ are pointers to pointers,

//variables that start with letters are not pointers

struct complex\_tag \*\_a,\*\_b,\*\_div, \*\_add, \*\_sub, \*\*\_add\_, \*\*\_sub\_, a, b, div = {0,0}, add = {1,2}, sub = {3,4};

int addSub, division;

//checking for sufficient number of arguments

if (argc < 5){

printf("Need more arguments\n");

return 0;

}

a.real = atof(argv[1]);

a.imaginary = atof(argv[2]);

b.real = atof(argv[3]);

b.imaginary = atof(argv[4]);

\_a = &a;

\_b = &b;

\_div = &div;

\_add = &add;

\_sub = &sub;

\_add\_ = &\_add;

\_sub\_ = &\_sub;

//display entered complex numbers

printf("\nENTERED COMPLEX NUMBERS:\n(%f) + (%f)i\n(%f) + (%f)i\n", a.real, a.imaginary, b.real, b.imaginary);

division = divide(\_a, \_b, \_div);

//display division result

if (division == 0){

printf("\nDIVISION RESULTS:\n(%f) + (%f)i\n\n", (\*\_div).real, (\*\_div).imaginary);

} else {

printf("\nDIVISION RESULTS:\nCan't divide, denomenator = 0\n\n");

}

Complex\_type mult = multiply(\*\_a,\*\_b);

//display multiplication result

printf("MULTIPLICATION RESULTS:\n(%f) + (%f)i\n\n", mult.real, mult.imaginary);

addSub = add\_subtract(\*\_a, \*\_b, \_add\_, \_sub\_);

//print error message if memory could not be allocated, else print addition and subtraction results

if (addSub == -1){

printf("Can't subtract or add\n");

} else {

printf("ADDITION RESULTS:\n(%f) + (%f)i\n\n", (\*\*\_add\_).real, (\*\*\_add\_).imaginary);

printf("SUBTRACTION RESULTS:\n(%f) + (%f)i\n\n", (\*\*\_sub\_).real, (\*\*\_sub\_).imaginary);

}

}

**makefile**

# MACRO definitions

CC = gcc

CFLAG = -std=c99 -Wall

# All Targets

all: operation

# operation depends on the files operation.o and operation\_functions.o

operation: operation.o operation\_functions.o

$(CC) $(CFLAG) -o operation operation.o operation\_functions.o

# operation.o depends on the source and header files

operation.o: operation.c operation\_functions.h

$(CC) $(CFLAG) -c operation.c

# operation\_functions.o depends on the source and header files

operation\_functions.o: operation\_functions.c operation\_functions.h

$(CC) $(CFLAG) -c operation\_functions.c

#clean the build inventory

clean:

rm -f \*.o operation

#test operation with values 1 2 3 4

test: operation

operation 1 2 3 4 > output\_file