**makefile**

#hash is only seen by the preprocessor and not the processor

#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\_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

# test cases

test: operation

operation 1.5 2 3 4

operation 3.3 0 7.8 0

operation 0 4 0 1

operation 2.3 0 0 9

operation 0 3 3.4 0

operation 0 0 1 2

operation 3 4 0 0

#Clean the build directory

clean:

rm -f \*.o

**operation\_functions.h**

#ifndef operation\_functions\_h

#define operation\_functions\_h

#include <stdio.h>

struct complex\_tag

{

double real;

double imaginary;

};

typedef struct

{

double real;

double imaginary;

}Complex\_type;

//part c)

Complex\_type multiplication(struct complex\_tag c1, struct complex\_tag c2);

//part d)

int division(struct complex\_tag \*pointerOne, struct complex\_tag \*pointerTwo, struct complex\_tag \*pointerThree);

//part e)

int sumAndDifference(struct complex\_tag c1, struct complex\_tag c2, struct complex\_tag \*\*pointerOne, struct complex\_tag \*\*pointerTwo);

#endif /\* operation\_function\_h \*/

**operation\_functions.c**

#include "operation\_functions.h"

#include <stdio.h>

#include <stdlib.h>

/\*NOTE: a+ib: a is the real part, b is the imaginary part\*/

//part c)

Complex\_type multiplication(struct complex\_tag c1, struct complex\_tag c2)

{

/\*computes the value for the product:

c1 \* c2 = (a1\*a2 - b1\*b2) + i (a2\*b1 + a1\*b2)\*/

Complex\_type multz;

multz.real = (c1.real \* c2.real) - (c1.imaginary \* c2.imaginary);

multz.imaginary = (c2.real \* c1.imaginary) + (c1.real \* c2.imaginary );

return multz; //returns the product

}

//part d)

int division(struct complex\_tag \*pointerOne, struct complex\_tag \*pointerTwo, struct complex\_tag \*pointerThree){

/\*if(a2\*a2 + b2\*b2) = 0, (the denominator is zero) return -2,\*/

if( ((pointerTwo->real \* pointerTwo->real)+(pointerTwo->imaginary \* pointerTwo->imaginary))==0 ){

return -2;

}

/\*pointerOne and pointerTwo point to the complex numbers\*/

/\*the value of pointerThree is the division result of the first two pointers. Computes the value for the division:

c1/c2 = (a1\*a2 + b1\*b2)/(a2\*a2 + b2\*b2) + i (a2\*b1 - a1\*b2)/(a2\*a2 + b2\*b2)\*/

//computes numerator for real:(a1\*a2 + b1\*b2)

(\*pointerThree).real = (pointerOne->real \* pointerTwo->real) + (pointerOne->imaginary \* pointerTwo->imaginary);

//computes denominator for real:(a2\*a2 + b2\*b2)

(\*pointerThree).real = pointerThree->real / ((pointerTwo->real \* pointerTwo->real) + (pointerTwo->imaginary \* pointerTwo->imaginary));

////

//computes numerator for imaginary: (a2\*b1 - a1\*b2)

(\*pointerThree).imaginary = (pointerTwo->real \* pointerOne->imaginary ) - (pointerOne->real \* pointerTwo->imaginary);

//computes denominator for imaginary: (a2\*a2 + b2\*b2)

(\*pointerThree).imaginary = pointerThree->imaginary /((pointerTwo->real \* pointerTwo->real) + (pointerTwo->imaginary \* pointerTwo->imaginary));

return 0;//operation successful, return 0

}

//part e)

int sumAndDifference(struct complex\_tag c1, struct complex\_tag c2, struct

complex\_tag \*\*pointerOne, struct complex\_tag \*\*pointerTwo)

{

/\*allocates memory for the two pointers\*/

(\*pointerOne) = malloc(sizeof(struct complex\_tag));

(\*pointerTwo) = malloc(sizeof(struct complex\_tag));

/\*Memory allocation operation unsuccessful, return -1\*/

if(pointerOne == NULL || pointerTwo == NULL){

return -1;

}

/\*value is the sum of the first two parameters:

c1+c2=(a1+a2)+i(b1+b2)\*/

(\*pointerOne)->real=(c1.real+c2.real);

(\*pointerOne)->imaginary=(c1.imaginary+c2.imaginary);

/\*value is the differenc ebtween the first two parameterrs:

c1-c2=(a1-a2)+i(b1-b2)\*/

(\*pointerTwo)->real=(c1.real - c2.real);

(\*pointerTwo)->imaginary=(c1.imaginary - c2.imaginary);

return 0;//successful return 0

}

**operation.c**

#include <stdio.h>

#include <stdlib.h>

#include "operation\_functions.h"

////////////////////////PROTOTYPES//////////////////////

void printOut(struct complex\_tag \*numba);

///////////////////////////////////////////////////////

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

{

/\*CHECK IF THERE IS THE CORRECT NUMBER OF ARGUMENTS BEFORE PRECEDING\*/

if (argc !=5){

printf("Incorrect number of arguments \n");

return -1; //unsuccessful

}

/\*Declares two variables of type complex\_t. The value of these two

variables will be initialized using the command-line

arguments as 4 separate values, two for each variable. \*/

struct complex\_tag compOne;

struct complex\_tag compTwo;

/\*declares structure variables and pointers to store the results

of the functions\*/

Complex\_type mult; //store the product

struct complex\_tag quotient;//store the division

//store sum and difference. also pointers to the structures

//and pointers to the pointers

struct complex\_tag add, sub;

struct complex\_tag \*ptr1 = &add, \*ptr2 = &sub;

struct complex\_tag \*\*add1 = &ptr1, \*\*sub1 = &ptr2;

/\*Initialization: since argv is an array of characters, use

array to float function: atof() to convert\*/

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

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

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

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

mult = multiplication(compOne, compTwo);//invokes the multiplication function to initialize the value

/\*Computations which will determine the values to be printed\*/

/\*Return value for division function: if value returned is negative, print

the error message and the printOut function will NOT be called\*/

int div = division(&compOne, &compTwo, &quotient);

/\*Return value for sum and difference function: if value returned is negative,

print the error message and the printOut function will NOT be called\*/

int sd = sumAndDifference(compOne, compTwo, add1, sub1);

/\*print the entered complex numbers and the results of the functions\*/

printf("First complex number: ");

printOut(&compOne);

printf("Second complex number: ");

printOut(&compTwo);

printf("The product: ");

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

printf("The division: ");

if (div == 0){ //If division successful, call printOut

printOut(&quotient);

}

else{ //else print division was unsuccessful

printf("Error, can't divide by zero \n");

}

if (sd == 0){ //sum and difference successful

printf("The sum: ");

printOut(ptr1);

printf("The difference: ");

printOut(ptr2);

}

else{ //unsuccessful, print error message

printf("Error, couldn't allocate memory for pointer \n");

}

printf("\n\n");

return 0; //success

}

/\*function for printing\*/

void printOut(struct complex\_tag \*numba)

{

double realz = numba->real;

double imaginaryz = numba->imaginary;

printf("%f + i %f \n", realz, imaginaryz);

}

**TEST CASES**

obelix.gaul.csd.uwo.ca[78]% make test

gcc -std=c99 -Wall -c operation\_functions.c

gcc -std=c99 -Wall -o operation operation.o operation\_functions.o

operation 1.5 2 3 4

First complex number: 1.500000 + i 2.000000

Second complex number: 3.000000 + i 4.000000

The product: -3.500000 + i 12.000000

The division: 0.500000 + i 0.000000

The sum: 4.500000 + i 6.000000

The difference: -1.500000 + i -2.000000

operation 3.3 0 7.8 0

First complex number: 3.300000 + i 0.000000

Second complex number: 7.800000 + i 0.000000

The product: 25.740000 + i 0.000000

The division: 0.423077 + i 0.000000

The sum: 11.100000 + i 0.000000

The difference: -4.500000 + i 0.000000

operation 0 4 0 1

First complex number: 0.000000 + i 4.000000

Second complex number: 0.000000 + i 1.000000

The product: -4.000000 + i 0.000000

The division: 4.000000 + i 0.000000

The sum: 0.000000 + i 5.000000

The difference: 0.000000 + i 3.000000

operation 2.3 0 0 9

First complex number: 2.300000 + i 0.000000

Second complex number: 0.000000 + i 9.000000

The product: 0.000000 + i 20.700000

The division: 0.000000 + i -0.255556

The sum: 2.300000 + i 9.000000

The difference: 2.300000 + i -9.000000

operation 0 3 3.4 0

First complex number: 0.000000 + i 3.000000

Second complex number: 3.400000 + i 0.000000

The product: 0.000000 + i 10.200000

The division: 0.000000 + i 0.882353

The sum: 3.400000 + i 3.000000

The difference: -3.400000 + i 3.000000

operation 0 0 1 2

First complex number: 0.000000 + i 0.000000

Second complex number: 1.000000 + i 2.000000

The product: 0.000000 + i 0.000000

The division: 0.000000 + i 0.000000

The sum: 1.000000 + i 2.000000

The difference: -1.000000 + i -2.000000

operation 3 4 0 0

First complex number: 3.000000 + i 4.000000

Second complex number: 0.000000 + i 0.000000

The product: 0.000000 + i 0.000000

The division: Error, can't divide by zero

The sum: 3.000000 + i 4.000000

The difference: 3.000000 + i 4.000000