ASSIGNMENT #3 – COMPLEX DATA STRUCTURES

CS3540/03 SYSTEM PROGRAMMING WITH C AND LINUX

SPRING 2015

Baturay Daylak

bdaylak@students.kennesaw.edu

January 28th, 2015

Department of Computer Science

College of Science and Mathematics

Kennesaw State University

Assignment Description:

CS3540 Assignment No. 3

(Due Jan 28, 2015)

Using the structure defined below, *struct ComplexNum*, develop a C program that performs complex addition, subtraction, multiplication, and division of the complex values of the elements of array x with the complex values of the elements in array y.

```
struct ComplexNum {
   double realpart;
   double imagpart;
};
```

Indicate clearly all the commands used, compiler, editor used, etc. Work on Linux (use either the CS3 server or your own installation of Linux)

1. Provide a session log on Linux using the 'script' command
2. Use the 'tee' command on Linux to provide redirection of the output data (results).

Solution Details:

All solutions are developed on Debian 7 running within VirtualBox 4.3.20 hosted by Windows 8.1 environment. Debian 7's default GNU Compiler is used to compile the source files. Source file is named *assignment3.c.* Command used to compile this file is:

gcc -Wall assignment3.c -o assignment3

Command used to run the program to direct its output to a file named *output.txt*:

./assignment3 | tee output.txt

Solution to Problem 1)

Problem states that the structure ComplexNum to be utilized in a program in which fundamental algebraic operations on complex numbers are also defined. These operations are defined as distinct functions to promote flexibility and error detection, if any. Only one function handles the array operations. One of the parameters is this function is to specify which operation to be performed on the two other arrays, which are passed to the function by pointers.

Source:

```
// Program: CS3540/03 Assignment 3 - ComplexNumber Operations
// Author: Baturay Daylak


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

const ARRAYSIZE = 10; // ARRAY SIZE FOR ALL ARRAYS WHICH ARE DEFINED IN THIS PROGRAM

//COMPLEX NUMBER STRUCTURE
struct ComplexNum {
            double realpart;
            double imagpart;
};

// BEGIN DEFINITION OF ALGEBRAIC OPERATIONS ON COMPLEX NUMBERS
struct ComplexNum add(struct ComplexNum a, struct ComplexNum b)
{
            struct ComplexNum result;
            result.realpart = a.realpart + b.realpart;
            result.imagpart = a.imagpart + b.imagpart;
            return result;
}

struct ComplexNum substract(struct ComplexNum a, struct ComplexNum b)
{
            struct ComplexNum result;
            result.realpart = a.realpart - b.realpart;
            result.imagpart = a.imagpart - b.imagpart;
            return result;
```

```
}

struct ComplexNum divide(struct ComplexNum a, struct ComplexNum b)
{
            struct ComplexNum result;
            result.realpart = a.realpart / b.realpart;
            result.imagpart = a.imagpart / b.imagpart;
            return result;
}

struct ComplexNum multiply(struct ComplexNum a, struct ComplexNum b)
{
            struct ComplexNum result;
            result.realpart = a.realpart * b.realpart;
            result.imagpart = a.imagpart * b.imagpart;
            return result;
}// END DEFINITION OF ALGEBRAIC OPERATIONS ON COMPLEX NUMBERS



// BEGIN ARRAY OPERATIONS FUNCTION
//
// This function returns a pointer to a ComplexNum typed memory field.
// Array allocation is done through malloc function of stdlib, using a constant for array size.
// First argument is a ComplexNum typed array
// Second argument is also a ComplexNum typed array
// Third argument is opcode, which are following
//        0 for addition
//        1 for substraction
//        2 for division
//        3 for multiplication
//
```

```
// This way, one function handles all four operations interchangebly.
// Tidy.
struct ComplexNum * arrayops(struct ComplexNum a[], struct ComplexNum b[], int opcode)
{

		struct ComplexNum * results;
		int i;
		results = malloc(ARRAYSIZE * sizeof(struct ComplexNum));
		for(i=0;i<ARRAYSIZE;i++)
			if(opcode == 0)
				results[i] = add(a[i], b[i]);
			else if(opcode == 1)
				results[i] = substract(a[i], b[i]);
			else if(opcode == 2)
				results[i] = divide(a[i], b[i]);
			else if(opcode == 3)
				results[i] = multiply(a[i], b[i]);

		return results;
}// END ARRAY OPERATIONS FUNCTION



// BEGIN PROGRAM
int main()
{

		struct ComplexNum a[ARRAYSIZE], b[ARRAYSIZE];
		int i;
```

```
	struct ComplexNum *radd, *rsub, *rdiv, *rmul;

	printf("\n\nProgram: Assignment 3.1 Complex number operations\nAuthor: Baturay Daylak\n\n Input data:\n");

	// This loop is to fill up the ComplexNum arrays.
	// Because it's boring to manually create those.
	// ARRAYSIZE can be changed at the top of the definitions, and this will continue to work.
	// By default, 10 seems to be enough.
	for(i=0; i<ARRAYSIZE; i++)
	{

		a[i].realpart = i+100;
		a[i].imagpart = 50 - i;

		b[i].realpart = i + 5;
		b[i].imagpart = 60 + i;

		printf("Array 1 Element %d) Real: %lf\t Imaginary: %lf\n", i+1, a[i].realpart, a[i].imagpart);
		printf("Array 2 Element %d) Real: %lf\t Imaginary: %lf\n", i+1, b[i].realpart, b[i].imagpart);

	}

	// For opcode definitions, refer to the function definition inline doc.
	radd = arrayops(a, b, 0); //Addition
	rsub = arrayops(a, b, 1); //Substraction
	rdiv = arrayops(a, b, 2); //Division
	rmul = arrayops(a, b, 3); //Multiplication

	printf("\n\nAddition results\n");
```

```
	for(i=0; i<ARRAYSIZE; i++)
		printf("\t%d) Real: %lf\t Imaginary: %lf\n", i+1, radd[i].realpart, radd[i].imagpart);

	printf("\nSubstraction results\n");
		for(i=0; i<ARRAYSIZE; i++)
			printf("\t%d) Real: %lf\t Imaginary: %lf\n", i+1, rsub[i].realpart, rsub[i].imagpart);

	printf("\nMultiplication results\n");
		for(i=0; i<ARRAYSIZE; i++)
			printf("\t%d) Real: %lf\t Imaginary: %lf\n", i+1, rmul[i].realpart, rmul[i].imagpart);

	printf("\nDivision results\n");
		for(i=0; i<ARRAYSIZE; i++)
			printf("\t%d) Real: %lf\t Imaginary: %lf\n", i+1, rdiv[i].realpart, rdiv[i].imagpart);

	printf("\n\n"); //leave spaces on CLI  after finishing output so that output can be seen easily.

	return 0;

} // END PROGRAM
```

Output

```
Program: Assignment 3.1 Complex number operations
Author: Baturay Daylak

 Input data:
Array 1 Element 1) Real: 100.000000        Imaginary: 50.000000
Array 2 Element 1) Real: 5.000000             Imaginary: 60.000000
Array 1 Element 2) Real: 101.000000        Imaginary: 49.000000
Array 2 Element 2) Real: 6.000000             Imaginary: 61.000000
Array 1 Element 3) Real: 102.000000        Imaginary: 48.000000
Array 2 Element 3) Real: 7.000000             Imaginary: 62.000000
Array 1 Element 4) Real: 103.000000        Imaginary: 47.000000
Array 2 Element 4) Real: 8.000000             Imaginary: 63.000000
Array 1 Element 5) Real: 104.000000        Imaginary: 46.000000
Array 2 Element 5) Real: 9.000000             Imaginary: 64.000000
Array 1 Element 6) Real: 105.000000        Imaginary: 45.000000
Array 2 Element 6) Real: 10.000000           Imaginary: 65.000000
Array 1 Element 7) Real: 106.000000        Imaginary: 44.000000
Array 2 Element 7) Real: 11.000000           Imaginary: 66.000000
Array 1 Element 8) Real: 107.000000        Imaginary: 43.000000
Array 2 Element 8) Real: 12.000000           Imaginary: 67.000000
Array 1 Element 9) Real: 108.000000        Imaginary: 42.000000
Array 2 Element 9) Real: 13.000000           Imaginary: 68.000000
Array 1 Element 10) Real: 109.000000      Imaginary: 41.000000
Array 2 Element 10) Real: 14.000000        Imaginary: 69.000000


Addition results
                1) Real: 105.000000          Imaginary: 110.000000
                2) Real: 107.000000          Imaginary: 110.000000
                3) Real: 109.000000          Imaginary: 110.000000
                4) Real: 111.000000          Imaginary: 110.000000
```

```
5) Real: 113.000000          Imaginary: 110.000000
6) Real: 115.000000          Imaginary: 110.000000
7) Real: 117.000000          Imaginary: 110.000000
8) Real: 119.000000          Imaginary: 110.000000
9) Real: 121.000000          Imaginary: 110.000000
10) Real: 123.000000       Imaginary: 110.000000
```

Substraction results

```
1) Real: 95.000000            Imaginary: -10.000000
2) Real: 95.000000            Imaginary: -12.000000
3) Real: 95.000000            Imaginary: -14.000000
4) Real: 95.000000            Imaginary: -16.000000
5) Real: 95.000000            Imaginary: -18.000000
6) Real: 95.000000            Imaginary: -20.000000
7) Real: 95.000000            Imaginary: -22.000000
8) Real: 95.000000            Imaginary: -24.000000
9) Real: 95.000000            Imaginary: -26.000000
10) Real: 95.000000          Imaginary: -28.000000
```

Multiplication results

```
1) Real: 500.000000          Imaginary: 3000.000000
2) Real: 606.000000          Imaginary: 2989.000000
3) Real: 714.000000          Imaginary: 2976.000000
4) Real: 824.000000          Imaginary: 2961.000000
5) Real: 936.000000          Imaginary: 2944.000000
6) Real: 1050.000000       Imaginary: 2925.000000
7) Real: 1166.000000       Imaginary: 2904.000000
8) Real: 1284.000000       Imaginary: 2881.000000
9) Real: 1404.000000       Imaginary: 2856.000000
10) Real: 1526.000000     Imaginary: 2829.000000
```

```
Division results
                1) Real: 20.000000            Imaginary: 0.833333
                2) Real: 16.833333            Imaginary: 0.803279
                3) Real: 14.571429            Imaginary: 0.774194
                4) Real: 12.875000            Imaginary: 0.746032
                5) Real: 11.555556            Imaginary: 0.718750
                6) Real: 10.500000            Imaginary: 0.692308
                7) Real: 9.636364              Imaginary: 0.666667
                8) Real: 8.916667              Imaginary: 0.641791
                9) Real: 8.307692              Imaginary: 0.617647
                10) Real: 7.785714            Imaginary: 0.594203
```