

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:

```
00000001 // Program: CS3540/03 Assignment 3 - ComplexNumber Operations
00000002 // Author: Baturay Daylak
00000003
00000004
00000005 #include <stdio.h>
00000006 #include <stdlib.h>
00000007
00000008 const ARRAYSIZE = 10; // ARRAY SIZE FOR ALL ARRAYS WHICH ARE DEFINED
IN THIS PROGRAM
00000009
00000010 //COMPLEX NUMBER STRUCTURE
00000011 struct ComplexNum {
00000012     double realpart;
00000013     double imagpart;
00000014 };
00000015
00000016 // BEGIN DEFINITION OF ALGEBRAIC OPERATIONS ON COMPLEX NUMBERS
00000017 struct ComplexNum add(struct ComplexNum a, struct ComplexNum b)
00000018 {
00000019     struct ComplexNum result;
00000020     result.realpart = a.realpart + b.realpart;
00000021     result.imagpart = a.imagpart + b.imagpart;
00000022     return result;
00000023 }
00000024
00000025 struct ComplexNum subtract(struct ComplexNum a, struct ComplexNum b)
00000026 {
00000027     struct ComplexNum result;
00000028     result.realpart = a.realpart - b.realpart;
00000029     result.imagpart = a.imagpart - b.imagpart;
00000030     return result;
```

```
00000031 }
00000032
00000033 struct ComplexNum divide(struct ComplexNum a, struct ComplexNum b)
00000034 {
00000035     struct ComplexNum result;
00000036     result.realpart = a.realpart / b.realpart;
00000037     result.imagpart = a.imagpart / b.imagpart;
00000038     return result;
00000039 }
00000040
00000041 struct ComplexNum multiply(struct ComplexNum a, struct ComplexNum b)
00000042 {
00000043     struct ComplexNum result;
00000044     result.realpart = a.realpart * b.realpart;
00000045     result.imagpart = a.imagpart * b.imagpart;
00000046     return result;
00000047 }// END DEFINITION OF ALGEBRAIC OPERATIONS ON COMPLEX NUMBERS
00000048
00000049
00000050
00000051 // BEGIN ARRAY OPERATIONS FUNCTION
00000052 //
00000053 // This function returns a pointer to a ComplexNum typed memory field.
00000054 // Array allocation is done through malloc function of stdlib, using a
00000055 // constant for array size.
00000056 // First argument is a ComplexNum typed array
00000057 // Second argument is also a ComplexNum typed array
00000058 // Third argument is opcode, which are following
00000059 //      0 for addition
00000060 //      1 for subtraction
00000061 //      2 for division
00000062 //      3 for multiplication
00000063 //
```

```
00000063 // This way, one function handles all four operations interchangeably.
00000064 // Tidy.
00000065 struct ComplexNum * arrayops(struct ComplexNum a[], struct ComplexNum
b[], int opcode)
00000066 {
00000067
00000068     struct ComplexNum * results;
00000069     int i;
00000070     results = malloc(ARRAYSIZE * sizeof(struct ComplexNum));
00000071     for(i=0;i<ARRAYSIZE;i++)
00000072         if(opcode == 0)
00000073             results[i] = add(a[i],
b[i]);
00000074         else if(opcode == 1)
00000075             results[i] =
subtract(a[i], b[i]);
00000076         else if(opcode == 2)
00000077             results[i] = divide(a[i],
b[i]);
00000078         else if(opcode == 3)
00000079             results[i] =
multiply(a[i], b[i]);
00000080
00000081     return results;
00000082 }// END ARRAY OPERATIONS FUNCTION
00000083
00000084
00000085
00000086 // BEGIN PROGRAM
00000087 int main()
00000088 {
00000089
00000090     struct ComplexNum a[ARRAYSIZE], b[ARRAYSIZE];
00000091     int i;
```

```
00000092      struct ComplexNum *radd, *rsub, *rdiv, *rmul;
00000093
00000094      printf("\n\nProgram: Assignment 3.1 Complex number
operations\nAuthor: Baturay Daylak\n\n Input data:\n");
00000095
00000096      // This loop is to fill up the ComplexNum arrays.
00000097      // Because it's boring to manually create those.
00000098      // ARRAYSIZE can be changed at the top of the definitions,
and this will continue to work.
00000099      // By default, 10 seems to be enough.
00000100      for(i=0; i<ARRAYSIZE; i++)
00000101      {
00000102
00000103          a[i].realpart = i+100;
00000104          a[i].imagpart = 50 - i;
00000105
00000106          b[i].realpart = i + 5;
00000107          b[i].imagpart = 60 + i;
00000108
00000109          printf("Array 1 Element %d) Real: %lf\t
Imaginary: %lf\n", i+1, a[i].realpart, a[i].imagpart);
00000110          printf("Array 2 Element %d) Real: %lf\t
Imaginary: %lf\n", i+1, b[i].realpart, b[i].imagpart);
00000111
00000112      }
00000113
00000114      // For opcode definitions, refer to the function
definition inline doc.
00000115      radd = arrayops(a, b, 0); //Addition
00000116      rsub = arrayops(a, b, 1); //Substraction
00000117      rdiv = arrayops(a, b, 2); //Division
00000118      rmul = arrayops(a, b, 3); //Multiplication
00000119
00000120      printf("\n\nAddition results\n");
```

```
00000121         for(i=0; i<ARRAYSIZE; i++)
00000122                     printf("\t%d) Real: %lf\t Imaginary:
00000123 %lf\n", i+1, radd[i].realpart, radd[i].imagpart);
00000124
00000125         printf("\nSubstraction results\n");
00000126         for(i=0; i<ARRAYSIZE; i++)
00000127                     printf("\t%d) Real: %lf\t
00000128 Imaginary: %lf\n", i+1, rsub[i].realpart, rsub[i].imagpart);
00000129
00000130         printf("\nMultiplication results\n");
00000131         for(i=0; i<ARRAYSIZE; i++)
00000132                     printf("\t%d) Real: %lf\t
00000133 Imaginary: %lf\n", i+1, rmul[i].realpart, rmul[i].imagpart);
00000134
00000135         printf("\nDivision results\n");
00000136         for(i=0; i<ARRAYSIZE; i++)
00000137                     printf("\t%d) Real: %lf\t
00000138 Imaginary: %lf\n", i+1, rdiv[i].realpart, rdiv[i].imagpart);
00000139
00000140         printf("\n\n"); //leave spaces on CLI  after finishing
00000141         output so that output can be seen easily.
00000142
00000143         return 0;
00000144 } // 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