Lab No 5 & 6

Try C-programming features in the context of embedded systems

# Objective

1. Introduction to C-programming
2. Using the MPLAB IDE, Visual Studio and DEV

# C BASICS

# Values and Statements

* A **statement** is a unit of code that does something
  + a basic building block of a program.
* An **expression** is a statement that has a value, for instance,
  + a number,
  + a string,
  + the sum of two numbers, etc. 4 + 2, x - 1, and
  + "hello, world!\n"

are all expressions.

* Not every statement is an expression e.g.
  + #include <stdio.h>

**Operators**

* Used to perform calculations
* Act on expressions to form a new expression
* Types
  + Mathematical
  + Relational
  + Logical
  + Bitwise
  + Conditional
* ++ (arithmetic perspective)
  + - -- (arithmetic perspective)
    - ! (logical perspective) etc
* Binary Operators
  + - Arithmetic

– +, -, \*, /, %

* + - Relational

– >, <. >=, <=, ==, !=

* + - Logical

– &&, ||

**Data Types**

**Built-in Data types**

* Character types (such as **char**)
* Integer types (such as **int**)
* Floating-point types (such as **double**)

**User-defined Data types**

* Enumeration types for representing specific sets of values (**enum**)
* Pointer types (such as **int \***)
* Array types (such as **char[]**)
* Structures

**void**

* used to signify the absence of type information

Hello World

7

// A Hello World program #include <stdio.h >

int main(void)

{

printf("hello, world!\n");

return 0;

}

**Example**

char c = 'c';

// c is 'c' i.e., 0x63 int i = 1;

// i has value 1 i = 3.14;

// i has value 3 double pi = i;

// pi has value 3.0 unsigned char c = -1;

// assuming 8-bit chars, c has value 255

signed char c2

// assuming 8-bit

= 256;

chars, the value of c2

is undefined or 0 in little endian

**Functions in C**

* There can be only one function of a given name
* Function argument type checking is optional
  + Use a compiler option that makes it compulsory
* There are no references (and therefore no pass-by-reference)
  + pass a pointer
* There are no member functions
* There is an alternative function definition syntax

**Example**

* Our function

int square(int x)

{

return x \* x;

}

is an example of

Return\_type function\_name(Parameter list) // (type name, etc.)

{

// use each parameter in code return some\_value;// of Return\_type

}

35

**Conditional Compilation**

* #if
* #endif
* #ifdef
* #ifndef

#ifdef cplusplus

// in C++ #else

/\* in C \*/ #endif

**Type Aliasing by typedef Keyword**

A declaration prefixed by the keyword typedef declares a new name for the type rather than a new variable of the given type

typedef double gpa;

typedef char ascii;

typedef int int32;

typedef short int16;

**ARRAYS**

**One-Dimensional Arrays**

* An *element* of an array is accessed using the **array name** and an **index or subscript***, for example:* y[5] which can be used like a variable
* In VC++, the subscripts always start with 0 and increment by 1, so

y[5] is the **sixth** element

* The name of the array is the **address** of the first element and the subscript is the **offset**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |  |

y[0] y[1] y[2] … y[k-1] y[k] y[k+1] … y[98] y[99]

**Initializing Arrays**

* Arrays can be initialized at the time they are declared.
* Examples:

double taxrate[3] = { 0.15, 0.25, 0.3 };

char list[5] = { 'h', 'e', 'l', 'l', 'o' };

double vec[10] = { 0.0 }; //assigns zero to all 10 elements int s[] = { 5,0,-5 }; // the size of s is 3

63

**C-style strings**

In C a string (called a C-string or a C-style string in C++ literature) is a zero-terminated array of characters

char\* p = "asdf"; char s[] = "asdf";

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 'a' | 's' | 'd' | 'f' | 0 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 'a' | 's' | 'd' | 'f' | 0 |

**POINTERS**

* A **pointer** is a variable that holds the *address* of a memory location
* If a variable ***p*** holds the address of another variable ***q***, then ***p*** is said to point to ***q***
* If ***q*** is a variable at location 40 in memory, then ***p***
* would have the value 40 (***q***’s address)

How to declare a pointer variable

* Pointer variables are declared using an asterisk **\***

before the pointer name

**int a, b, \*ptr;**

* **ptr** is a pointer to a memory location that can store

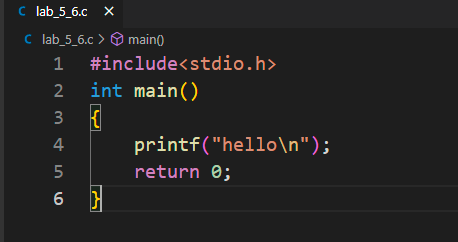
an integer

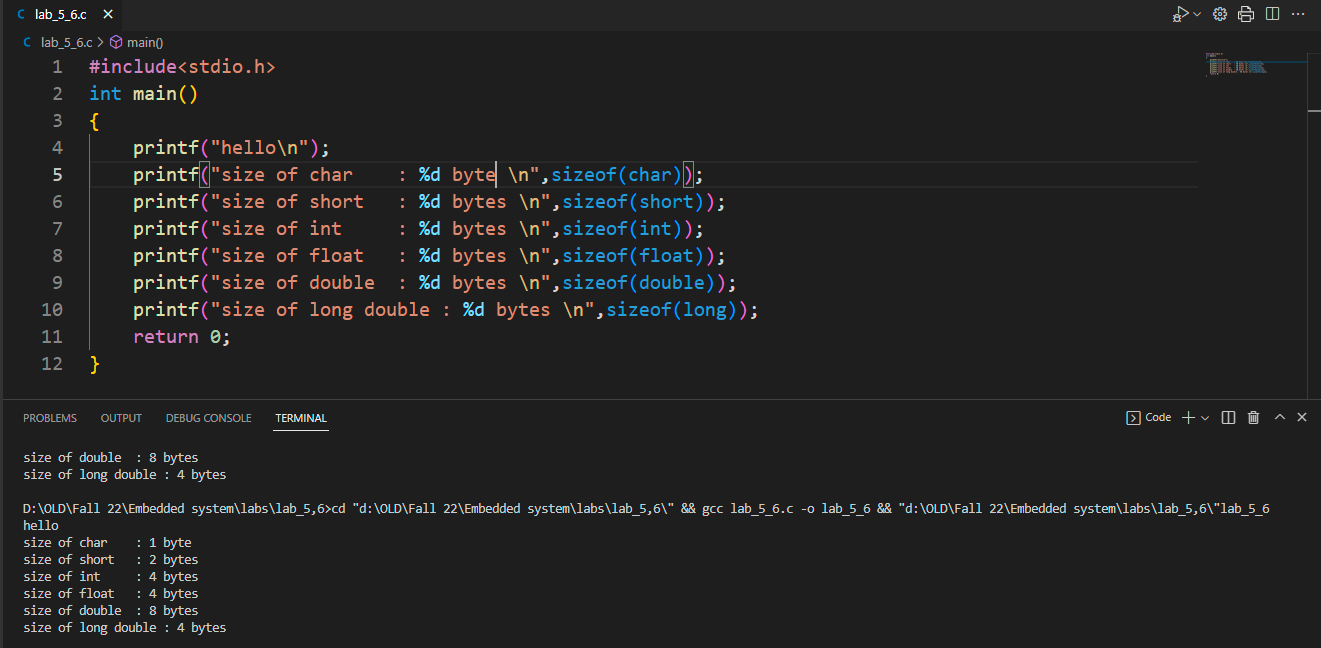
**Stages of Debugging**

* That can’t happen
* That doesn’t happen on my machine
* That shouldn’t happen
* Why does that happen?
* Oh, I see
* How did that ever work?

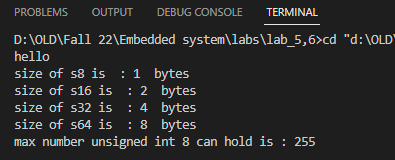
# Lab Tasks

1. Write simplest C-code i.e. **main(){}**

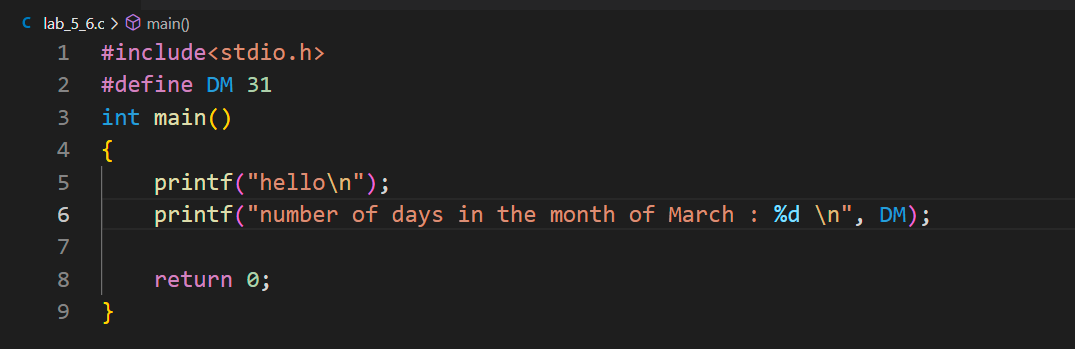


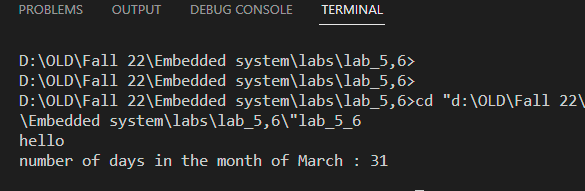
1. Print sizes of basic data types i.e. char, short, int etc.. using **sizeof.**
2. Giving custom name to data type like s8, u8, s16 .. using **typedef**.

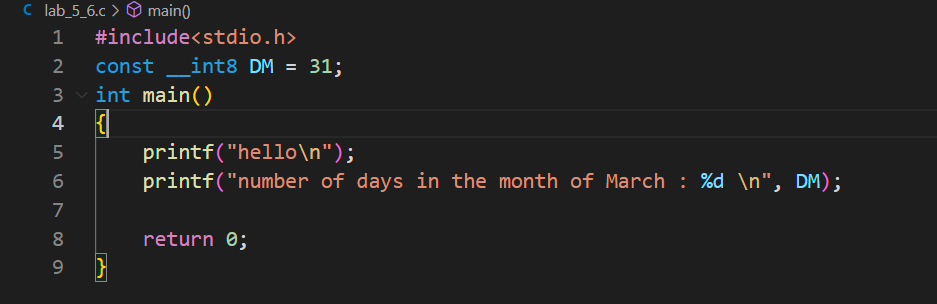
**Output:**



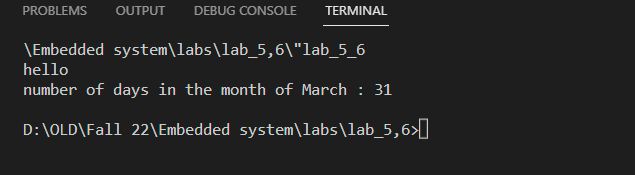
1. Use pre-processor directive **#define** to define number of days in the month of March.



**Output:**

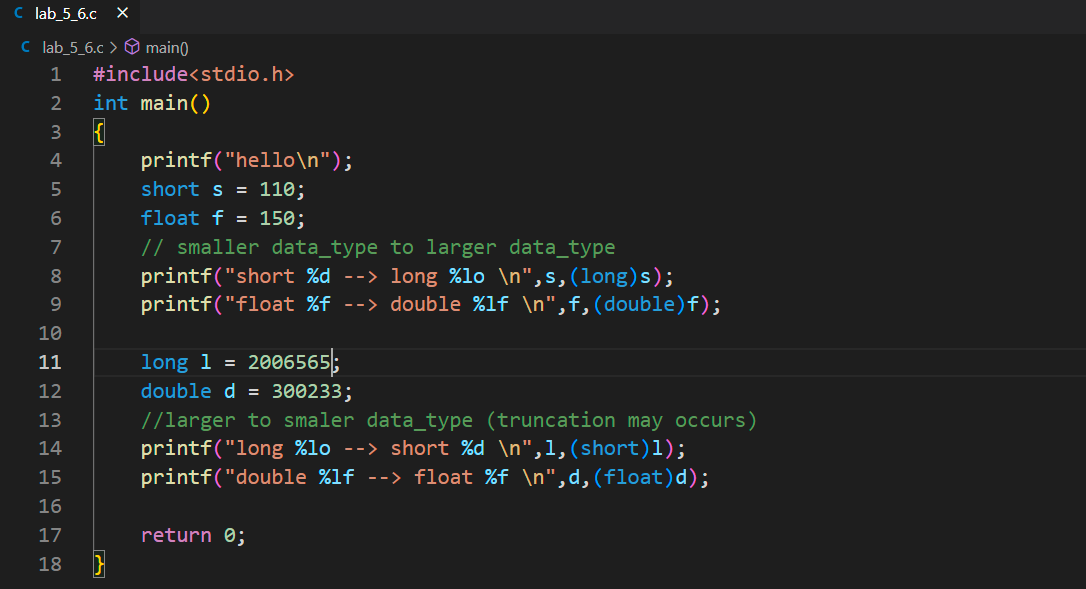
1. Use type qualifier const to define number of days in the month of March.

**Output:**

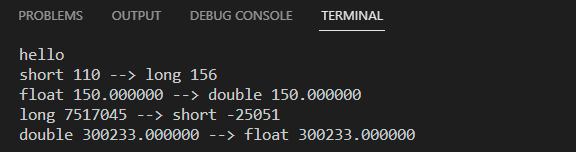


# Typecasting

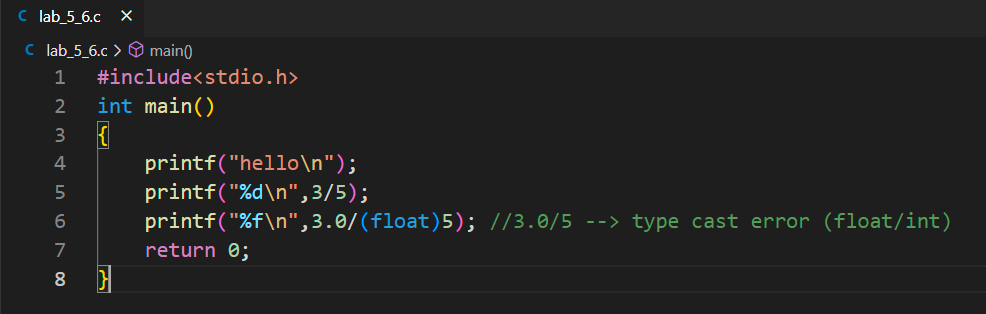
* 1. Copy

1. a **short** variable to **long** variable,
2. a **float** variable to **double** variable
   1. Copy
      1. a **long** variable to **short** variable.
      2. a **double** variable to **float** variable

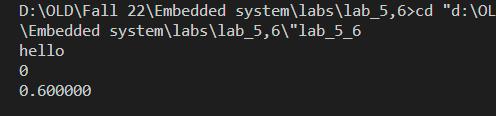
**Output:**



* 1. Perform division of following;
     1. 3/5
     2. 3.0/5



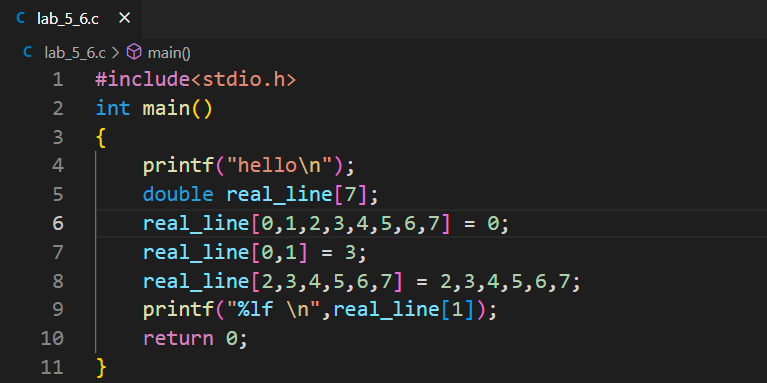
**Output:**



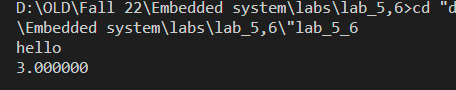
# Arrays

Make an array **double real\_line[7]** .

* 1. Initialize the whole array with zero.
  2. Initialize first 2 elements with 3.
  3. Initialize every element with its index position without using loop.

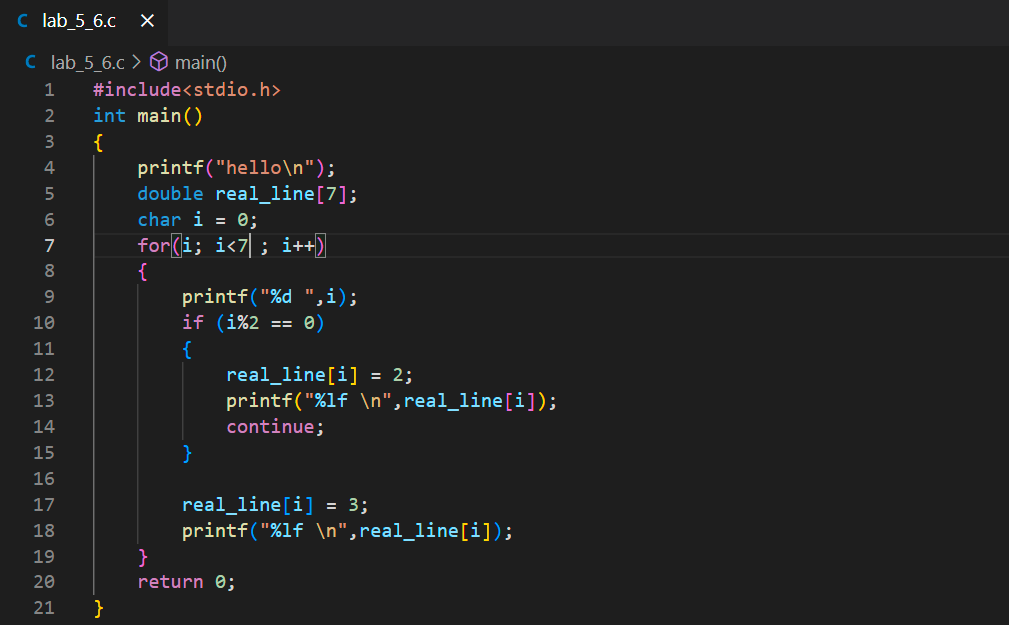


**Output:**

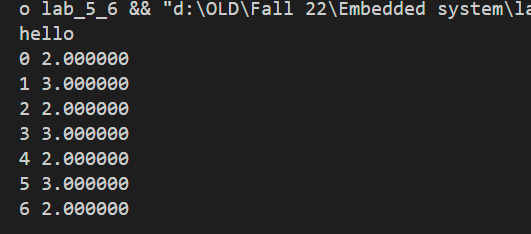


# Loops (for)

* 1. Use **for** loop to do task-2c.
  2. Calculate the sum of all the members of **real\_line** array.
  3. For array declared in task 2, set elements at even position to 2 and elements at odd position to 3.

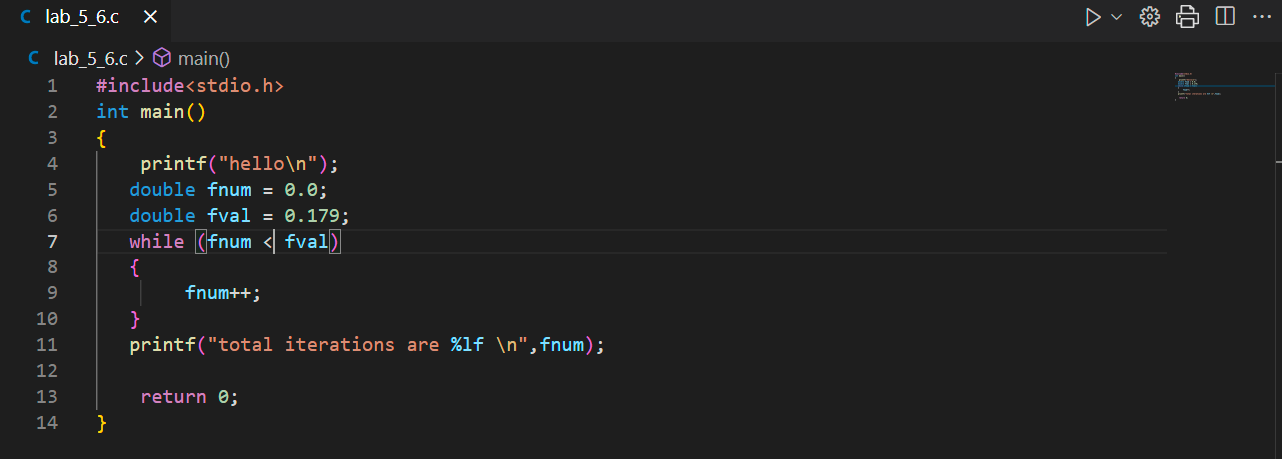


**Output:**

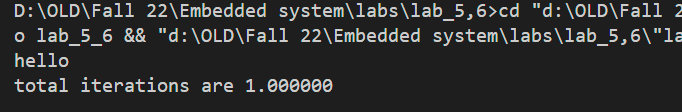


# Loops (while)

* 1. Declare variable **double fnum = 0.0** and **double fval = 0.179**;
  2. Use **while** loop to calculate how many loop iterations (at minimum) it would take to reach the value of



**Output:**



1. **Array of Structures**
   1. Make structure to define complex data type, having members **re** and **im** of **double** type. Use

**typedef** to name it **complex**.

* 1. A line **line1** is defined by 5 points having co-ordinates (0,0), (1,2), (2,3), (3,4) and (4,5). Use array of complex structure to store these points.
  2. Another line **line2** is defined by 5 points having co-ordinates (0,0), (1,-2), (2,-3), (3,-4) and (4,-5). Use array of complex structure to store these points.
  3. Define following complex functions;

# complex cadd( complex c1, complex c2);

* + - 1. **complex csub( complex c1, complex c2);**
      2. **complex cmul( complex c1, complex c2);**
      3. **complex cdiv( complex c1, complex c2);**
      4. **double creal(complex c);**
      5. **double cimag(complex c);**
      6. **complex comp( double re, double im);**
      7. **double cmag(complex c);**
      8. **double cangle(complex c);**
      9. **comp cadd\_real( complex c, double re);**
      10. **comp cmul\_real( complex c, double re);**
      11. **comp cdiv\_real( complex c, double re);**

**CODE:**

*#include<stdio.h>*

*typedef struct*

*{*

*double re;*

*double imag;*

*}complex;*

*complex cadd(complex , complex );*

*complex csub(complex , complex );*

*complex cmul(complex , complex );*

*complex cdiv(complex , complex );*

*int main()*

*{*

*printf("hello\n");*

*complex line1[5],line2[4],result;*

*line1[0].re = 0;*

*line1[0].imag= 0;*

*line1[1].re = 1;*

*line1[1].imag= 2;*

*line1[2].re = 2;*

*line1[2].imag= 3;*

*line1[3].re = 3;*

*line1[3].imag= 4;*

*line1[4].re = 4;*

*line1[4].imag= 5;*

*// struct 2*

*line2[0].re = 0;*

*line2[0].imag= 0;*

*line2[1].re = 1;*

*line2[1].imag= -2;*

*line2[2].re = 2;*

*line2[2].imag= -3;*

*line2[3].re = 3;*

*line2[3].imag= -4;*

*line2[4].re = 4;*

*line2[4].imag= -5;*

*char i = 0;*

*for (i; i<5 ; i++)*

*{*

*printf("%d \t",i);*

*result = cadd(line1[i],line2[i]);*

*printf("cadd: re %lf imag %lf \n",result.re,result.imag);*

*}*

*printf("\n\n");*

*i=0;*

*for (i; i<5 ; i++)*

*{*

*printf("%d \t",i);*

*result = csub(line1[i],line2[i]);*

*printf("csub: re %lf imag %lf \n",result.re,result.imag);*

*}*

*printf("\n\n");*

*i=0;*

*for (i; i<5 ; i++)*

*{*

*printf("%d \t",i);*

*result = cmul(line1[i],line2[i]);*

*printf("cmult: re %lf imag %lf \n",result.re,result.imag);*

*}*

*printf("\n\n");*

*i=0;*

*for (i; i<5 ; i++)*

*{*

*printf("%d \t",i);*

*result = cdiv(line1[i],line2[i]);*

*printf("cdiv: re %lf imag %lf \n",result.re,result.imag);*

*}*

*return 0;*

*}*

*complex cadd(complex c1,complex c2)*

*{*

*complex c3;*

*c3.re = c1.re + c2.re;*

*c3.imag= c1.imag + c2.imag;*

*return c3;*

*}*

*complex csub(complex c1 , complex c2)*

*{*

*complex c3;*

*c3.re = c1.re - c2.re;*

*c3.imag= c1.imag - c2.imag;*

*return c3;*

*}*

*complex cmul(complex c1 , complex c2)*

*{*

*complex c3;*

*c3.re = c1.re \* c2.re;*

*c3.imag= c1.imag \* c2.imag;*

*return c3;*

*}*

*complex cdiv(complex c1 , complex c2)*

*{*

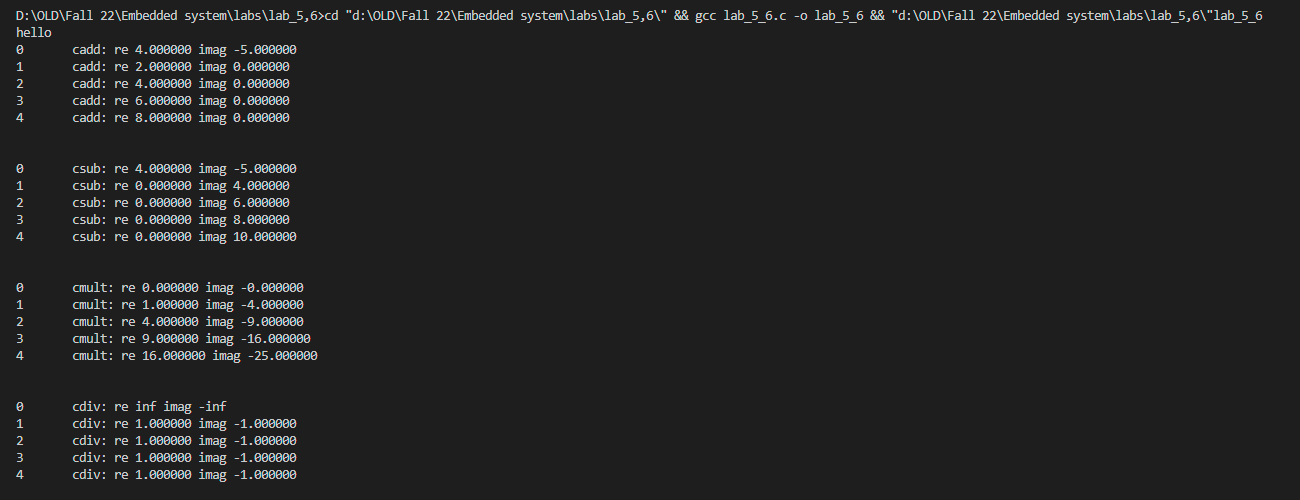
*complex c3;*

*c3.re = c1.re / c2.re;*

*c3.imag= c1.imag / c2.imag;*

*return c3;*

*}*



**Conclusion:**

|  |
| --- |
| **……………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………** |