**Pointers!**

Pointers are perhaps the most misused and misunderstood part of the C language. First, a pointer is type of data that 'points' to somewhere else in memory. Thus, a pointer to a character is declared as follows:

```
char * c_ptr;
```

This creates a variable that contains a pointer. The size of the pointer depends of the machine that the program is compiled on. In i386 Linux is 32 bits. Thus 4 bytes are reserved to 'point' to a character. It can also point to a bunch of characters or a string such as in the previous program. The catch is that since a pointer only saves enough room for an address where can the actual character string go? The programmer needs to create the data space for the character string by using `malloc(). malloc()` allocates memory for the program to use. Thus:

```
#include <stdio.h>
// include memory allocation header file
#include <stdlib.h>

int main(int argc, char * argv[])
{
        int * i_ptr_1;
        int * I_ptr_2;

        // allocate a room for an int &
        // cast -(int *)- the returned pointer from void * to int *
        // malloc returns void * and i_ptr is int * - so just do a cast
        // use sizeof() because the size maybe different on another machine

        i_ptr_1 = (int *)malloc(sizeof(int));
        i_ptr_2 = (int *)malloc(sizeof(int));

        // deference the pointer to get to the int data

        *i_ptr_1=45;
        *i_ptr_2=-60;

        printf("The sum of 45 and -60 is %s\n",add_two_numbers(*i_ptr_1,i_ptr_2));

        // release the memory
        free(i_ptr_1);
        free(i_ptr_2);
        return(0);
}
```

As stated before strings are arrays of characters and use pointers.

**Structures**

To create more complex data types, use structures. For example to create a complex number type use this structure:

```
struct complex_number {
        double real;   // Real part
        double imag;   // Imaginary
};  // Semicolon here
```

This declares the structure whose elements are the real part and imaginary part. To use it define a variable of type `struct complex_number`:

```
struct comple_number num;
```

The key work `struct` is use to declare the structure. Be sure to get the semicolon after the initial definition. To access the elements in the structure use a period (`.`). For example:

```
num.real = 4.2;
num.imag = 5.73;
```

To use pointers with structures, use the `*` to deference the pointer to get to the elements:

```
struct complex_number * num2;
// Use malloc to allocate memery for the structure.
(*num2).real = 34.59; // Awkward
```

or use the indirect reference operator (`->`).

```
num2->imag = 204.2385;        // Better
```

In the above example it is assumed that memory has be allocated with `malloc()` so that storage space is available for the elements of the structure.

Using the above complex number structure write four functions that add, subtract, multiply and divide complex numbers, and print a complex number. The functions should be prototype as follows:

```
int add_complex(struct complex_number * num1, struct complex_number num2);
int sub_complex(struct complex_number * num1, struct complex_number num2);
int mul_complex(struct complex_number * num1, struct complex_number num2);
int div_complex(struct complex_number * num1, struct complex_number num2);
int print_complex(struct complex_number num);
```

`num2` should be unaffected by the function but `num1` will contain the result of the operation. The pointer for the first argument make passing structures easier. C passes functions arguments on the stack - a temporary first in last out (FILO) data space - which is destroyed after the function returns. To get around this, pass the function a pointer to data. Thus the data being modified in the function is not on the stack (although the pointer to the data is) and will last for the life of the program. Use the above functions in a program and format your output. Place the definition of `complex_number` after all the `#include` but before `main()` so that all functions in your program will understand what `struct complex_number` is.