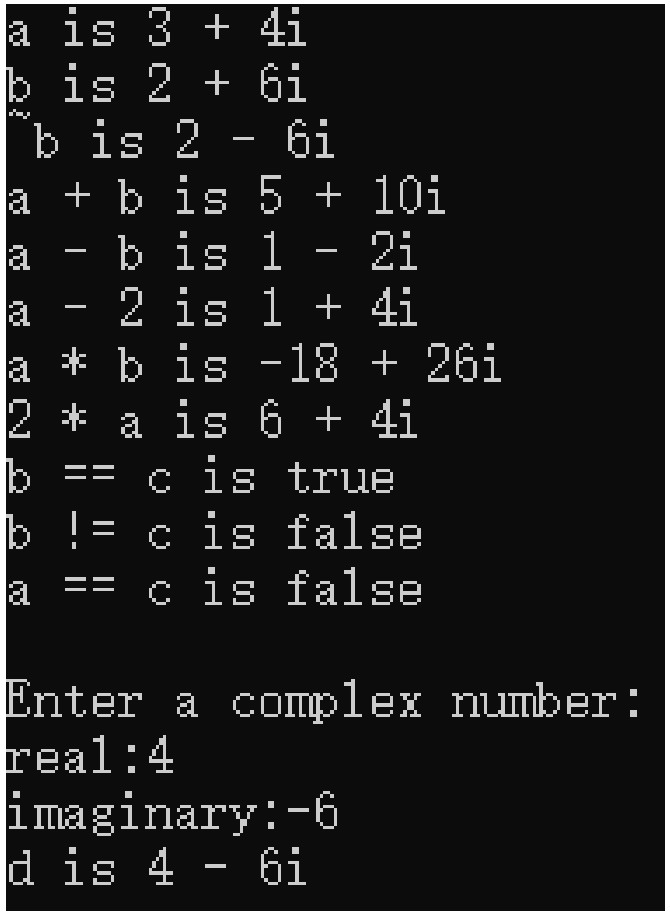
3 Exercises

1. Continue improving the Complex class and adding more operations for it, such as: -, *, ~, ==, != etc. Make the following program run correctly.

```cpp
#include <iostream>
#include "Complex.h"
using namespace std;

int main()
{
    Complex a( re: 3.0,  im: 4.0);
    Complex b( re: 2.0,  im: 6.0);

    cout << "a is " << a << endl;
    cout << "b is " << b << endl;
    cout << "~b is " << ~b << endl;
    cout << "a + b is " << a + b << endl;
    cout << "a - b is " << a - b << endl;
    cout << "a - 2 is " << a - 2 << endl;
    cout << "a * b is " << a * b << endl;
    cout << "2 * a is " << 2 * a << endl;

    Complex c = b;
    cout << "b == c is " << (b == c) << endl;
    cout << "b != c is " << (b != c) << endl;
    cout << "a == c is " << (a == c) << endl << endl;

    Complex d;
    cout << "Enter a complex number: " << endl;
    cin >> d;
    cout << "d is " << d << endl;

    return 0;
}
```

Note that you have to overload the << and >> operators. Use const whenever warranted.

A sample runs might look like this:



2. Design a class named **Number** that stores a number which is integer and private to the class. The default constructor should set the number to 0. Add another constructor that allows the caller to set the number. Finally, overload the prefix and postfix ++ and - - operators to make the following program run correctly.

```cpp
#include <iostream>
#include "Number.h"
using namespace std;

int main()
{
    Number n1( num: 20);
    Number n2 = n1++;
    cout << "n1 = " << n1 << endl;
    cout << "n2 = " << n2 << endl;

    Number n3 = n2--;
    cout << "n2 = " << n2 << endl;
    cout << "n3 = " << n3 << endl;

    Number n4 = ++n3;
    cout << "n3 = " << n3 << endl;
    cout << "n4 = " << n4 << endl;

    Number n5 = --n4;
    cout << "n4 = " << n4 << endl;
    cout << "n5 = " << n5 << endl;

    return 0;
}
```

A sample runs might look like this:

```
n1 = 21
n2 = 20
n2 = 19
n3 = 20
n3 = 21
n4 = 21
n4 = 20
n5 = 20
```

Note that you have to overload the << operators. Use const whenever warranted.

Hips:

Syntax for overloading postfix increment operator is as follows:

```
return-type operator ++(int)
{
    //Body of function
    ...
}
```

Syntax for overloading prefix increment operators is as follows:

```
return-type operator ++( )
{
    //Body of the function
    ...
}
```