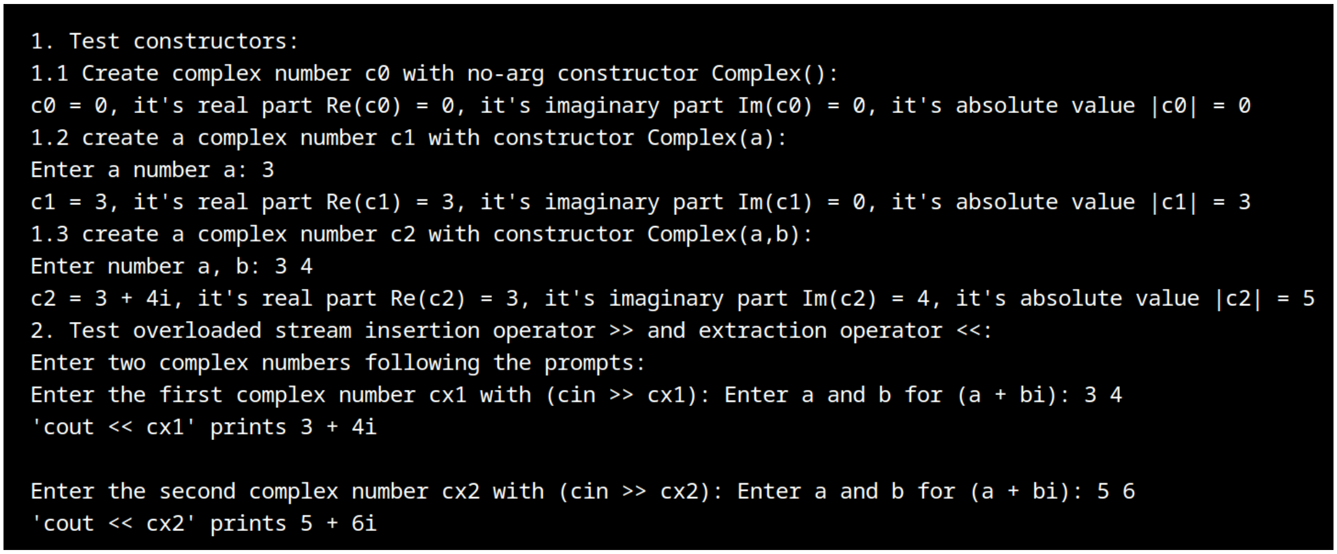
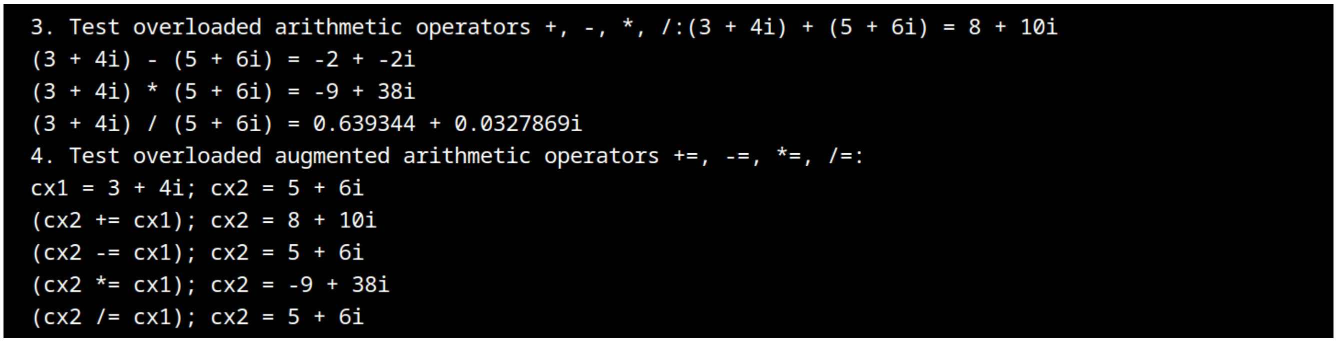
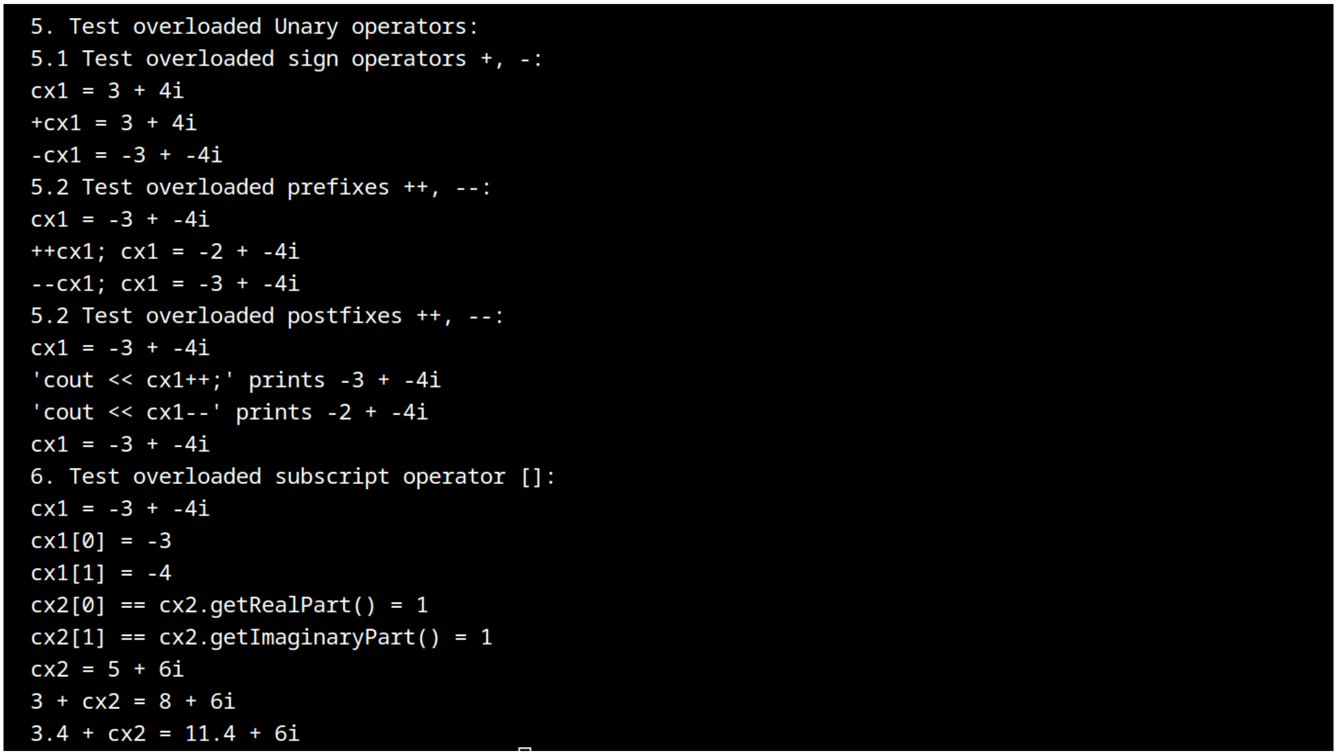


```
1. Test constructors:
1.1 Create complex number c0 with no-arg constructor Complex():
c0 = 0, it's real part Re(c0) = 0, it's imaginary part Im(c0) = 0, it's absolute value |c0| = 0
1.2 create a complex number c1 with constructor Complex(a):
Enter a number a: 3
c1 = 3, it's real part Re(c1) = 3, it's imaginary part Im(c1) = 0, it's absolute value |c1| = 3
1.3 create a complex number c2 with constructor Complex(a,b):
Enter number a, b: 3 4
c2 = 3 + 4i, it's real part Re(c2) = 3, it's imaginary part Im(c2) = 4, it's absolute value |c2| = 5
2. Test overloaded stream insertion operator >> and extraction operator <<:
Enter two complex numbers following the prompts:
Enter the first complex number cx1 with (cin >> cx1): Enter a and b for (a + bi): 3 4
'cout << cx1' prints 3 + 4i

Enter the second complex number cx2 with (cin >> cx2): Enter a and b for (a + bi): 5 6
'cout << cx2' prints 5 + 6i
```



```
3. Test overloaded arithmetic operators +, -, *, /:(3 + 4i) + (5 + 6i) = 8 + 10i
(3 + 4i) - (5 + 6i) = -2 + -2i
(3 + 4i) * (5 + 6i) = -9 + 38i
(3 + 4i) / (5 + 6i) = 0.639344 + 0.0327869i
4. Test overloaded augmented arithmetic operators +=, -=, *=, /=:
cx1 = 3 + 4i; cx2 = 5 + 6i
(cx2 += cx1); cx2 = 8 + 10i
(cx2 -= cx1); cx2 = 5 + 6i
(cx2 *= cx1); cx2 = -9 + 38i
(cx2 /= cx1); cx2 = 5 + 6i
```



```
5. Test overloaded Unary operators:
5.1 Test overloaded sign operators +, -:
cx1 = 3 + 4i
+cx1 = 3 + 4i
-cx1 = -3 + -4i
5.2 Test overloaded prefixes ++, --:
cx1 = -3 + -4i
++cx1; cx1 = -2 + -4i
--cx1; cx1 = -3 + -4i
5.2 Test overloaded postfixes ++, --:
cx1 = -3 + -4i
'cout << cx1++;' prints -3 + -4i
'cout << cx1--' prints -2 + -4i
cx1 = -3 + -4i
6. Test overloaded subscript operator []:
cx1 = -3 + -4i
cx1[0] = -3
cx1[1] = -4
cx2[0] == cx2.getRealPart() = 1
cx2[1] == cx2.getImaginaryPart() = 1
cx2 = 5 + 6i
3 + cx2 = 8 + 6i
3.4 + cx2 = 11.4 + 6i
```