

C++ OPERATOR OVERLOADING DESTRUCTOR

Problem Solving with Computers-II

C++

```
#include <iostream>
using namespace std;

int main(){
    cout<<"Hola Facebook\n";
    return 0;
}
```

Read the syllabus. Know what's required. Know how to get help.

Review Concepts from CS16

- Review basics of classes
 - Defining classes and declaring objects
 - Access specifiers: private, public
 - Different ways of initializing objects and when to use each:
 - Default constructor
 - Parametrized constructor
 - Parameterized constructor with default values
 - Initializer lists

Today's learning goals:

1. Operator overloading

- what is operator overloading?
- why/when would we need to overload operators?
- how to overload operators in C++ ?

2. Destructor:

- what is a destructor?
- why/when would we need one?
- how to implement a destructor?

How many objects of type Complex are created in main()?

```
int main(){
    Complex p;
    Complex *q = new Complex(2, 3);
    Complex w(10, -5);
    w.conjugate();
    w.print();
}
```

- A. One
- B. Two
- C. Three
- D. Four
- E. I am not sure . . .

```
class Complex
{
private:
    double real;
    double imag;
public:
    Complex(double re = 0, double im = 0);
    double getMagnitude() const;
    double getReal() const;
    double getImaginary() const;
    void print() const;
    void conjugate();
    void setReal(double r);
    void setImag(double r);
};
```

Fill in the blank to print the values of the object on the heap

```
int main(){
    Complex p;
    Complex *q = new Complex(2, 3);
    Complex w(10, -5);
    w.conjugate();
    w.print();
    
}
```

Desired output:

10 + 5j
2 + 3j

Review Constructor

- The constructor is a special method that is called right AFTER an object is created in memory (on the heap or stack)
- The compiler automatically generates a default constructor
- But you can implement a user-defined version

Operator overloading

```
int main(){
    Complex p;
    Complex *q = new Complex(2, 3);
    Complex w(10, -5);
    w.conjugate();
    w.print();
    q->print();
}
```

Before overloading the << operator

```
int main(){
    Complex p;
    Complex *q = new Complex(2, 3);
    Complex w(10, -5);
    w.conjugate();
    cout << w;
    cout << *q;
}
```

After overloading the << operator

```
cout << w;
```

Select any equivalent C++ statement:

```
w.operator<<(cout);
```

A

```
cout.operator<<(w);
```

B

```
operator<<(cout, w);
```

C

```
operator<<(cout, w);
```

Select the function declaration that best matches the above call

- A `void operator<<(ostream &out,
const Complex &c);`
- B `void Complex::operator<<(ostream &out);`
- C `Complex operator<<(ostream &out, Complex c);`

```
Complex w(1, 10), x(5, 1);  
cout<< w << x;
```

Select the function declaration that best matches the above call

A

```
void operator<<(ostream &out,  
                  const Complex &c);
```

B

```
Complex& operator<<(ostream &out,  
                        const Complex &c);
```

C

```
ostream& operator<<(ostream &out,  
                        const Complex &c);
```

Operator Overloading

We would like to be able to perform operations on two objects of the class using the following operators:

`<<`

`==`

`!=`

`+`

`-`

and possibly others

Constant pointers and pointers to constants

```
const char* p1;  
char* const p2;  
const char* const p3;
```

Constructor and Destructor

Every class has the following special methods:

- Constructor: Called right AFTER an object is created in memory
- Destructor: Called right BEFORE an object is deleted from memory

The compiler automatically generates default versions, but you can provide user-defined implementations

```
void foo(){
    Complex p(1, 2);
    Complex *q = new Complex(3, 4);
}
```

What is the output?

A. $1 + 2j$

B. $3 + 4j$

C. $1 + 2j$
 $3 + 4j$

D. None of the above

```
class Complex
{
private:
    double real;
    double imag;
public:
    Complex(double re = 0, double im = 0);
    ~Complex(){ print(); }
    double getMagnitude() const;
    double getReal() const;
    double getImaginary() const;
    void print() const;
    void conjugate();
    void setReal(double r);
    void setImag(double r);
};
```

Summary

- Classes have member variables and member functions (method). An object is a variable where the data type is a class.
- You should know how to declare a new class type, how to implement its member functions, how to use the class type.
- Frequently, the member functions of an class type place information in the member variables, or use information that's already in the member variables.
- New functionality may be added using non-member functions, friend functions, and operator overloading
- If a class allocates data on the heap, then a user-defined destructor must be implemented to perform a clean-up procedure (de-allocate heap memory)

Next time

- Linked Lists and the rule of three