Data Structures and Object Oriented Programming

Lecture 15

Dr. Naveed Anwar Bhatti

Webpage: naveedanwarbhatti.github.io

Operator Overloading

Stream Insertion Operators

Class: Operator Overloading (<< and >>)

- Bitwise operator **>>** (right shift) and **<<** (left shift) are built-in operators in C/C++
- These two operators are overloaded in **system library (iostream)** for formatted input (cin) and output (cout) of built-in types.
- *cout* is an object of **ostream**
- *cin* is an object of **istream**

Class: Operator Overloading (<< and >>)

- Bitwise operator **>>** (right shift) and **<<** (left shift) are built-in operators in C/C++
- These two operators are overloaded in **system library (iostream)** for formatted input (cin) and output (cout) of built-in types.
- *cout* is an object of **ostream**
- *cin* is an object of **istream**

Very Important:
Only one object of ostream and istream can be created

Class: Operator Overloading (<< and >>)

- Overloading **<<** and **>>** make it extremely easy to output your class to screen and accept user input from the console

```
int main() {
    myclass foo(3, 1);
    cout << "My values are: ";
    foo.print();
    cout << "in centimeters";
    return 0;
}
```

Class: Operator Overloading (<< and >>)

- Overloading **<<** and **>>** make it extremely easy to output your class to screen and accept user input from the console

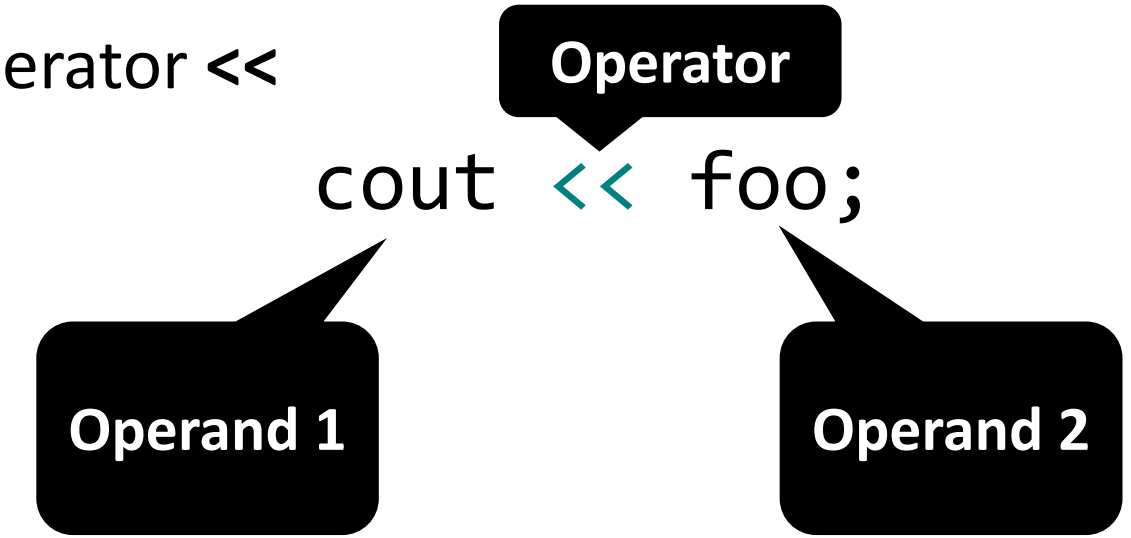
```
int main() {
    myclass foo(3, 1);
    cout << "My values are: ";
    foo.print();
    cout << "in centimeters";
    return 0;
}
```

```
int main() {
    myclass foo(3, 1);
    cout << "My values are: “<< foo << "in centimeters";
    return 0;
}
```

This would be much easier

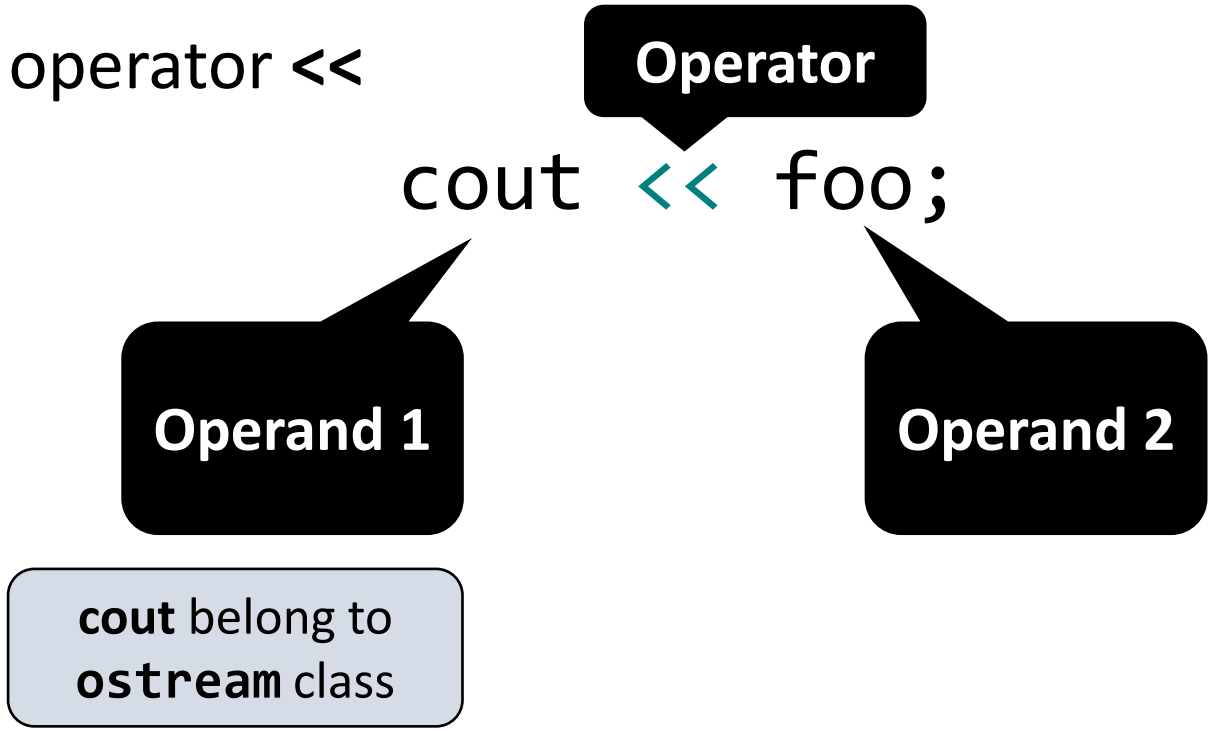
Class: Operator Overloading (<< and >>)

- Overloading operator <<



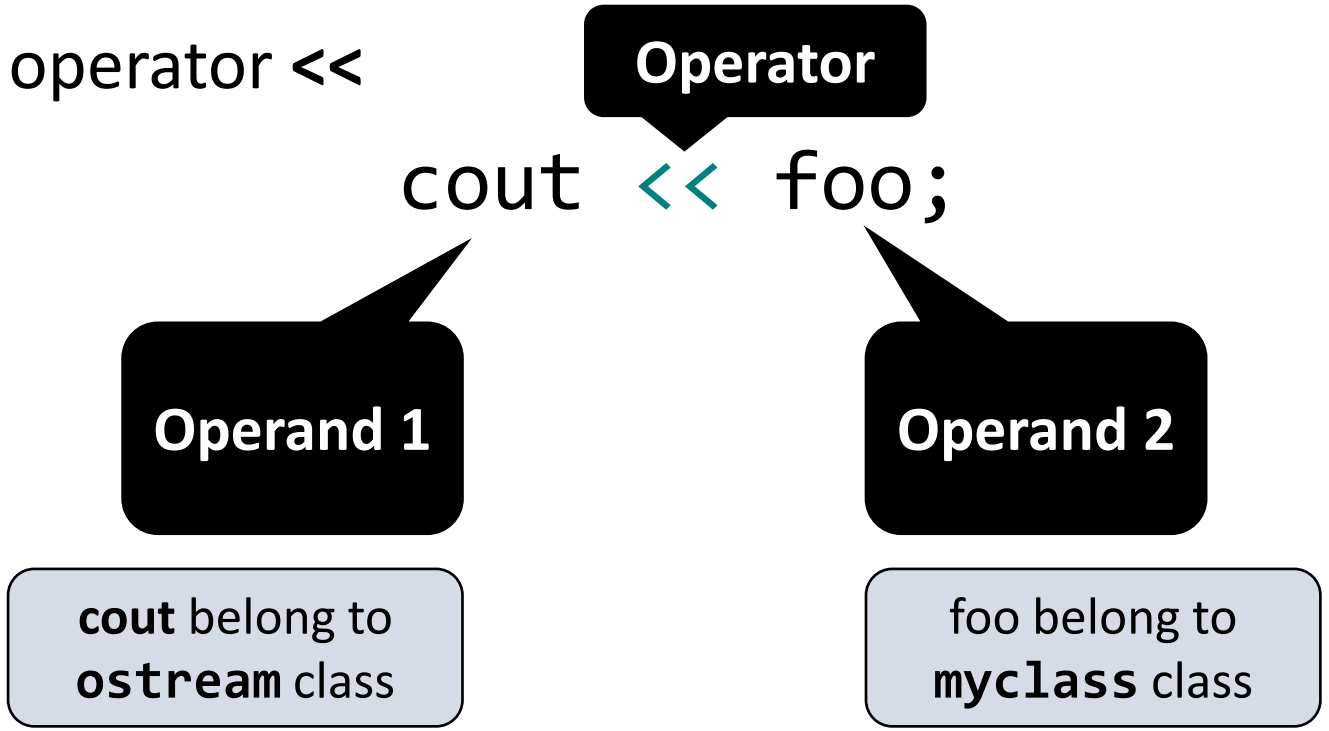
Class: Operator Overloading (<< and >>)

- Overloading operator <<



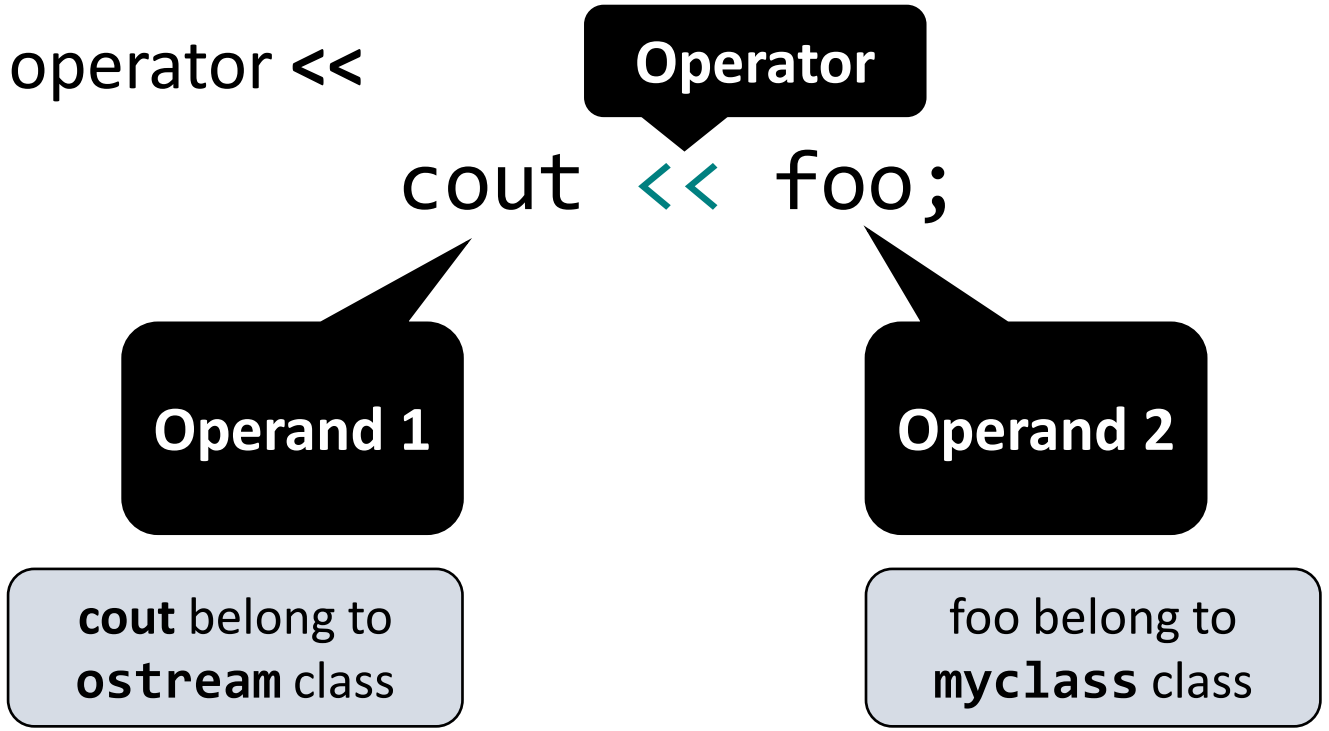
Class: Operator Overloading (<< and >>)

- Overloading operator <<



Class: Operator Overloading (<< and >>)

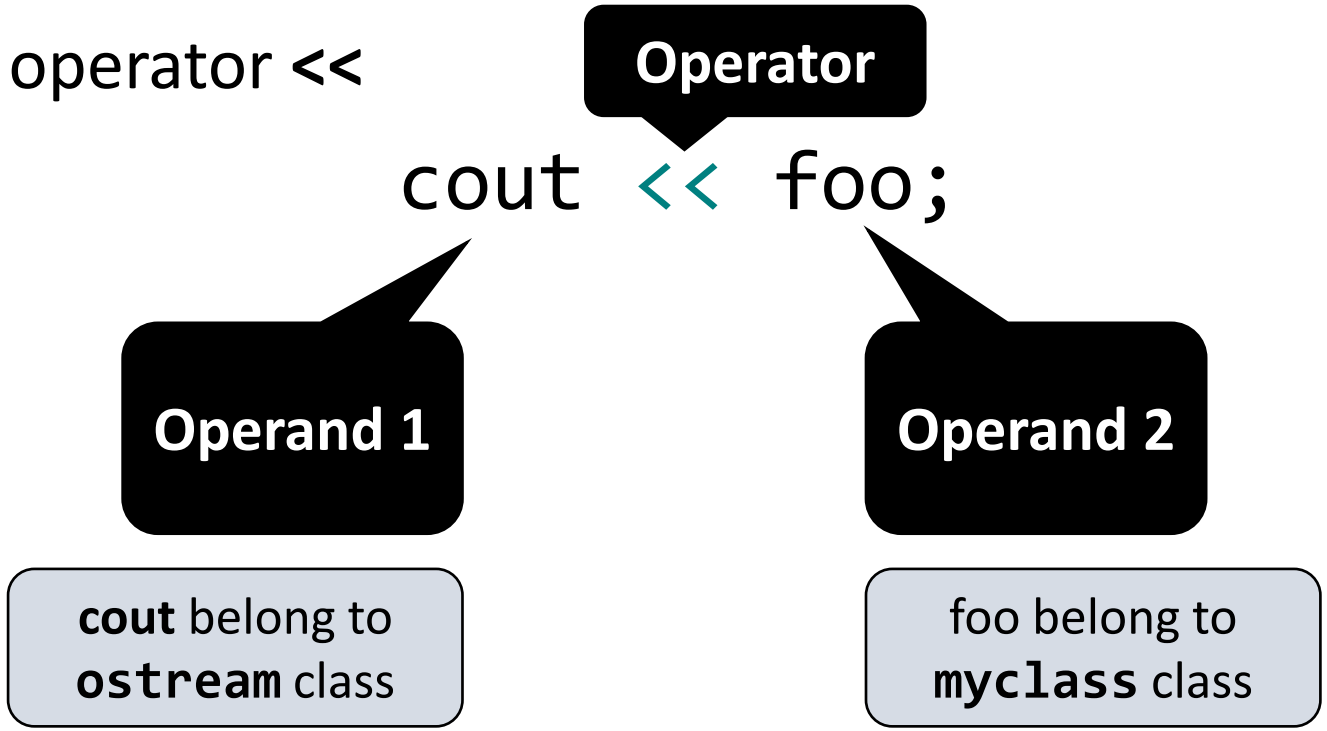
- Overloading operator <<



- Two methods:
  - **Member Function**
  - **Friend Function**

Class: Operator Overloading (<< and >>)

- Overloading operator <<



- Two methods:
  - **Member Function**
  - **Friend Function**

Incase of Member Function: We have to modifiy ostream class because cout is the first operand

Class: Operator Overloading (<< and >>)

- Overloading operator<<

```
class myclass {
    int x, y;
public:
    myclass() {};
    myclass(int, int);
    friend ostream& operator<<(ostream&, myclass);
};

myclass::myclass(int a, int b)
{
    x = a;
    y = b;
}

ostream& operator<<(ostream& os, myclass a)
{
    os << a.x << ',' << a.y;
    return os;
}
```

```
int main() {
    myclass foo(3, 1);
    cout << "My values are: “<<
    foo<< "in x,y cooridinates";
    return 0;
}
```

Class: Operator Overloading (<< and >>)

- Overloading operator<<

```
class myclass {
    int x, y;
public:
    myclass() {};
    myclass(int, int);
    friend ostream& operator<<(ostream&, myclass);
};

myclass::myclass(int a, int b)
{
    x = a;
    y = b;
}

ostream& operator<<(ostream& os, myclass a)
{
    os << a.x << ‘,' << a.y;
    return os;
}
```

Pass-by-Reference to avoid creating new object

```
int main() {
    myclass foo(3, 1);
    cout << "My values are: “<<
    foo<< "in x,y cooridinates";
    return 0;
}
```

Class: Operator Overloading (<< and >>)

Why are we returning `ostream&`?

- It allows us to “chain” output commands together
- And reference “&” to avoid creating new object

In case of `void:`

```
cout << foo;
```

No Error

but

```
cout << foo << “are x, y coordinates”;
```

Error

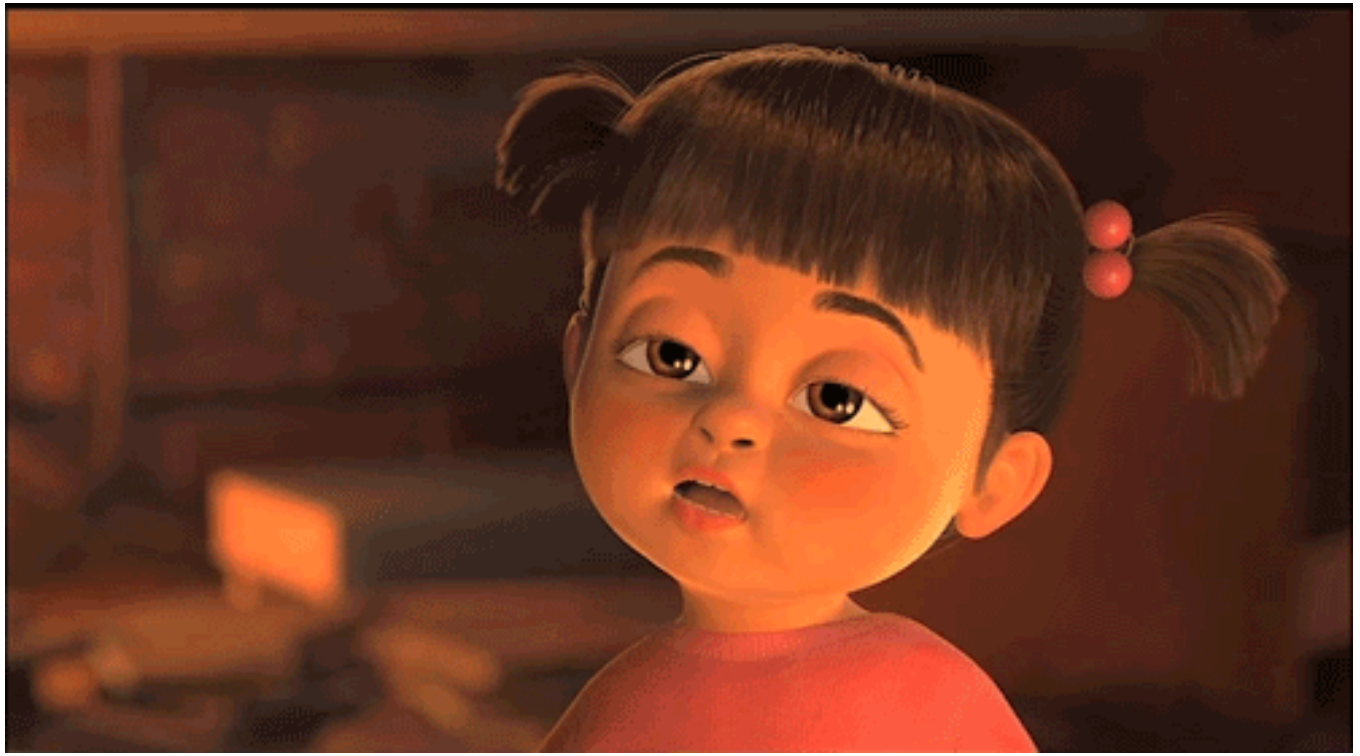
```
(cout << foo) << “are x, y coordinates”;

void << “are x, y coordinates”;
```

Reason

Class: Operator Overloading

Exercise: Overload ‘ >> ’ operator for same class using Friend Function.

Thanks a lot



If you are taking a Nap, **wake up**.....Lecture Over