Data Structures and Object Oriented Programming

Lecture 5

Dr. Naveed Anwar Bhatti

Webpage: naveedanwarbhatti.github.io

Object-Oriented Programming in C++

Operator Overloading

Class: Operator Overloading

- **Operator Basic**
  - **Operator:** An operator is a symbol that tells the compiler to perform specific mathematical, logical manipulations, or some other special operation.
  - Two Types: ***Binary Operator*** and ***Unary Operator***

- **Operator Overloading**
  - Refers to the multiple definitions of an operator
  - Arithmetic operator such as **+** and **/** are already overloaded in C/C++ for different built-in types.

Class: Operator Overloading

- Why we need it?
  - To make operators, i.e., **+**, **-**, **<**, **>**, etc., work for user defined data types/classes

- For example?

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

};
```

```
int main() {
    myclass foo(1, 1);
    myclass bar(1, 1);
    myclass result;
    result = foo + bar;
    return 0;
}
```

Error

Class: Operator Overloading

- Example

```
class myclass {
public:
    int x, y;
    myclass() {};
    myclass(int, int);
};

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

myclass operator+ (myclass param1, myclass param2) {
    myclass temp;
    temp.x = param1.x + param2.x;
    temp.y = param1.y + param2.y;
    return temp;
}
```

No data encapsulation

To overload operator +, the name of the operator function is operator+

```
int main() {
    myclass foo(3, 1);
    myclass bar(1, 2);
    myclass result;
    result = foo + bar;
    cout << result.x << ',' << result.y << '\n';
    return 0;
}
```

Microsoft Visual Studio Debug Console

```
4,3
```

Class: Operator Overloading

Two other methods which keeps “Data Encapsulation”:

- **Friend Function**
- **Member Function**

Class: Operator Overloading (using Friend Function)

- Example

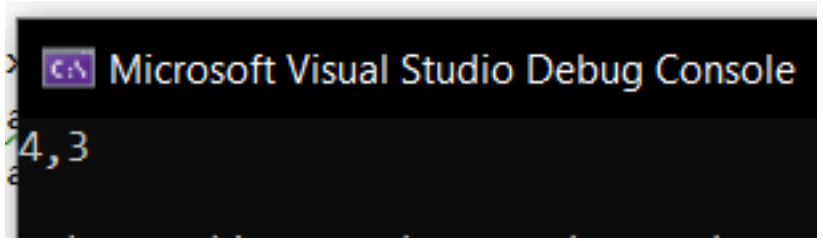
```
class myclass {
    int x, y;
public:
    myclass() {};
    myclass(int, int);
    friend myclass operator+ (myclass, myclass);
    void print();
};

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

void myclass::print() {
    cout << x << ',' << y << '\n';
}
```

```
myclass operator+ (myclass param1, myclass param2) {
    myclass temp;
    temp.x = param1.x + param2.x;
    temp.y = param1.y + param2.y;
    return temp;
}


int main() {
    myclass foo(3, 1);
    myclass bar(1, 2);
    myclass result;
    result = foo + bar;
    result.print();
    return 0;
}
```



Class: Operator Overloading (using Member Function)

- Example

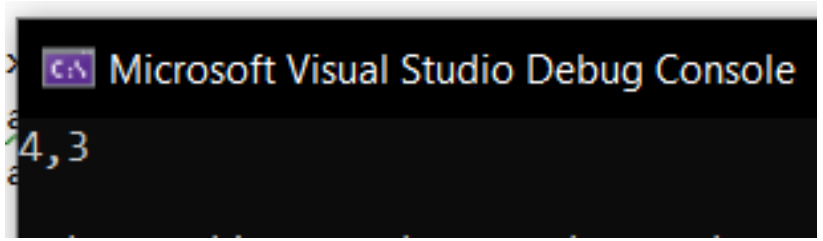
```
class myclass {
    int x, y;
public:
    myclass() {};
    myclass(int, int);
    myclass operator+ (myclass);
    void print();
};

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

void myclass::print() {
    cout << x << ',' << y << '\n';
}
```

```
myclass myclass::operator+ (myclass param2) {
    myclass temp;
    temp.x = x + param2.x;
    temp.y = y + param2.y;
    return temp;
}
```

```
int main() {
    myclass foo(3, 1);
    myclass bar(1, 2);
    myclass result;
    result = foo + bar;
    result.print();
    return 0;
}
```



Class: Operator Overloading

Overloadable Operators:

+	-	*	/	%	^
&	|	~	!	,	=
<	>	<=	>=	++	--
<<	>>	==	!=	&&	||
+=	-=	/=	%=	^=	&=
|=	*=	<<=	>>=	[]	()
->	->*	new	new []	delete	delete []

Class: Operator Overloading

Exercise: Overload ‘ * ’ operator for same class using both methods, i.e., Friend Function and Member Function.

Exercise: Overload ‘ - ’ operator for same class using both methods, i.e., Friend Function and Member Function.

Exercise: Overload ‘ / ’ operator for same class using both methods, i.e., Friend Function and Member Function.

Class: Operator Overloading (++ and - -)

- The operator ++ and –– have two forms : pre and post

```
int x = 6;
++x; // preincrement
x++; // postincrement
--x; // predecrement
x--; // postdecrement
```

- To overload the preincrement and predecrement operator, we use the declaration:

operator++();
operator--();

What about Postfix? Later

Class: Operator Overloading (++ and - -) – (Member Function)

- Example

```
class myclass {
    int x, y;
public:
    myclass() {};
    myclass(int, int);
    void operator++ ();
    void print();
};

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

void myclass::print() {
    cout << x << ',' << y << '\n';
}
```

```
void myclass::operator++ () {
    ++x;
    ++y;
}
```

```
int main() {
    myclass foo(3, 1);
    ++foo;
    foo.print();
    return 0;
}
```

Prefix version

Microsoft Visual Studio Debug Console

```
4,2
```

Class: Operator Overloading (++ and - -) – (Member Function)

- Example

```
class myclass {
    int x, y;
public:
    myclass() {};
    myclass(int, int);
    void operator++ (int);
    void print();
};

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

void myclass::print() {
    cout << x << ',' << y << '\n';
}
```

Compiler know its Postfix

```
void myclass::operator++ (int) {
    x++;
    y++;
}
```

```
int main() {
    myclass foo(3, 1);
    foo++;
    foo.print();
    return 0;
}
```

Microsoft Visual Studio Debug Console

```
4,2
```

Class: Operator Overloading (++ and - -)

- To overload the *preincrement* and *predecrement* operator, we use the declaration:

  operator++();
  operator--();

- To overload the *postincrement* and *postdecrement* operator, we use the declaration:

  operator++(int);
  operator--(int);

Class: Operator Overloading (++ and - -) – (Friend Function)

- Example

```
class myclass {
    int x, y;
public:
    myclass() {};
    myclass(int, int);
    friend void operator++ (myclass&);
    void print();
};

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

void myclass::print() {
    cout << x << ',' << y << '\n';
}
```

```
void operator++ (myclass& a) {
    ++a.x;
    ++a.y;
}
```

```
int main() {
    myclass foo(3, 1);
    ++foo;
    foo.print();
    return 0;
}
```

Microsoft Visual Studio Debug Console

```
4,2
```

Class: Operator Overloading (++ and - -) – (Friend Function)

- What will happen if I will write main() like this?

```
int main() {
    myclass foo(3, 1);
    myclass bar(0,0);
    bar = ++foo;
    bar.print();
    return 0;
}
```

Error

```
myclass operator++ (myclass& a) {
    myclass temp;
    temp.x=++a.x;
    temp.y=++a.y;
    return temp;
}
```

Microsoft Visual Studio Debug Console

```
4,2
```

Class: Operator Overloading (++ and - -)

Exercise: Overload ‘ -- ’ operator for same class using both methods, i.e., Friend Function and Member Function.

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 >>)

- 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 x,y cooridinates";
    return 0;
}
```

This would be much easier

```
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;
}
```

```
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 also allows us to “chain” output commands together

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 (++ and - -)

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

Thanks a lot

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