Object Oriented Programming

Lecture 11

Dr. Naveed Anwar Bhatti

Webpage: naveedanwarbhatti.github.io

Operator Overloading

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

- **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.

For example

The compiler probably calls the correct overloaded low level function for addition i.e:

```
// for integer addition:
operator+(int a, int b)

// for float addition:
operator+(float a, float b)
```

Operator Overloading

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

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; Error
    return 0;
}
```

Class: Operator Overloading

- Instead we have to do something like this?

```
class myclass {
    int x, y;
public:
    myclass(int a, int b)
    {
        x = a;
        y = b;
    }
   myclass add(myclass a)
    {
        myclass temp;
        temp.x= x + a.x
        temp.y= y + a.y;
        return temp;
    }
};
```

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

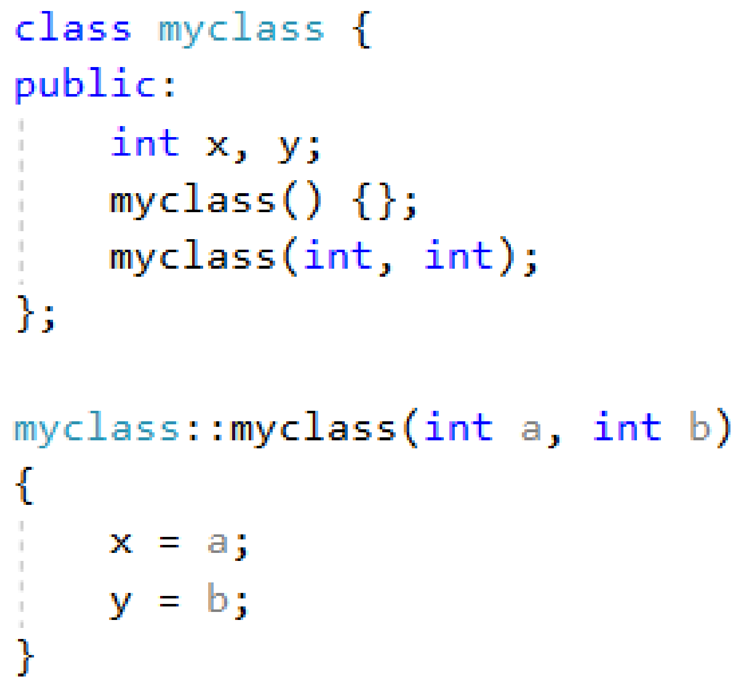
Correct

Operator Overloading

- If the mathematical expression is big:
  - Converting it to C++ code will involve complicated mixture of function calls
  - Less readable
  - Chances of human mistakes are very high
  - Code produced is very hard to maintain

Solution! = Operator Overloading

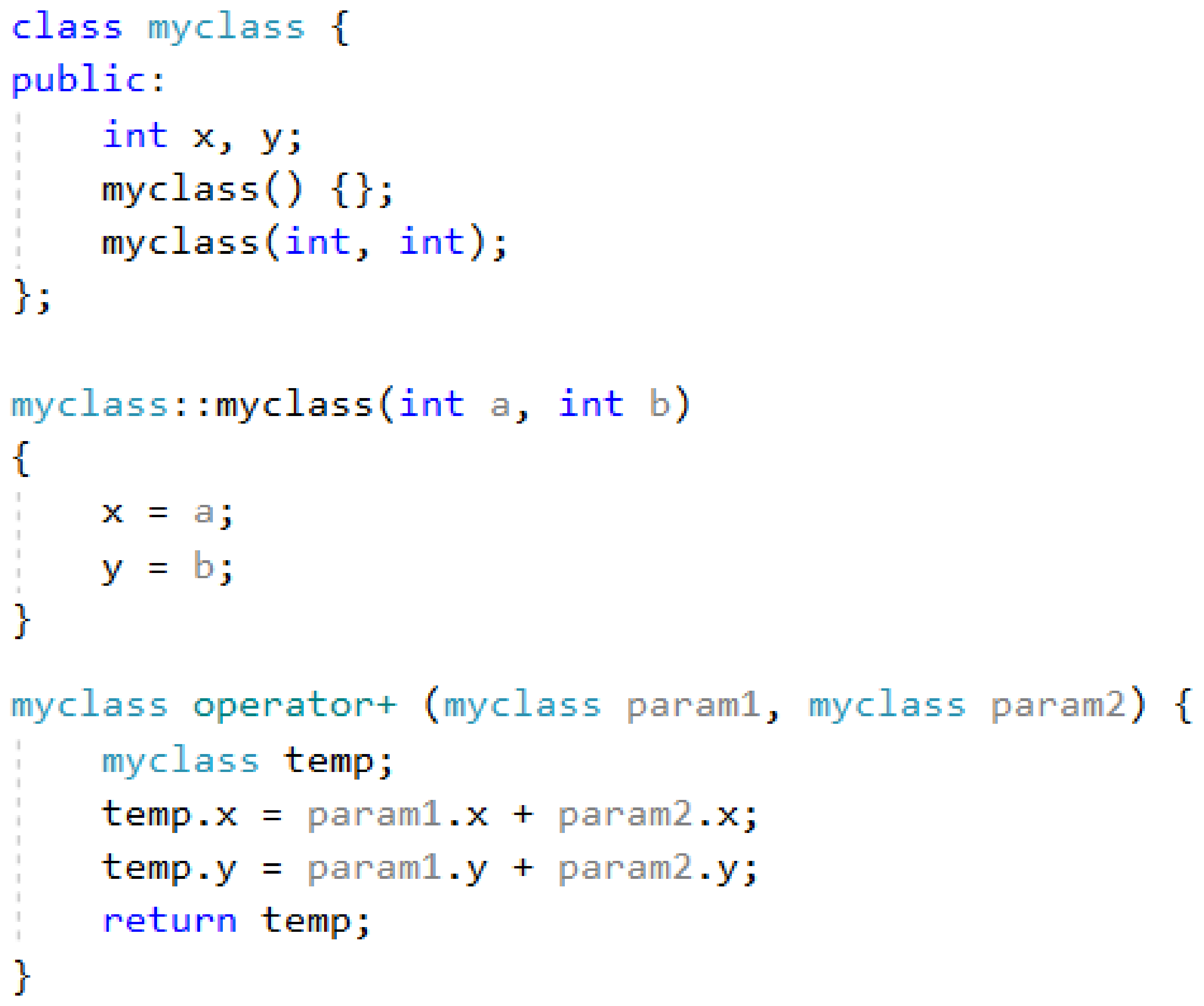
- Example



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

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

Solution! = 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;
}
```

Solution! = 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;
}
```

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

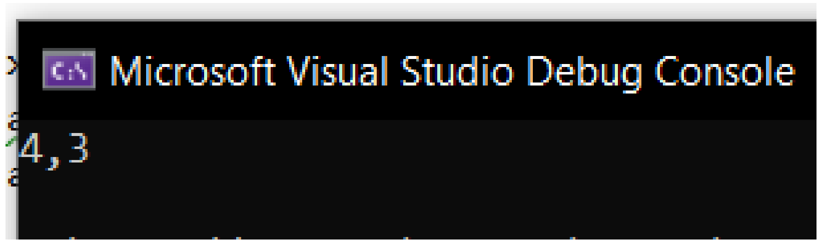
Solution! = Operator Overloading

- 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;
}
```

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
```

Solution! = Operator Overloading

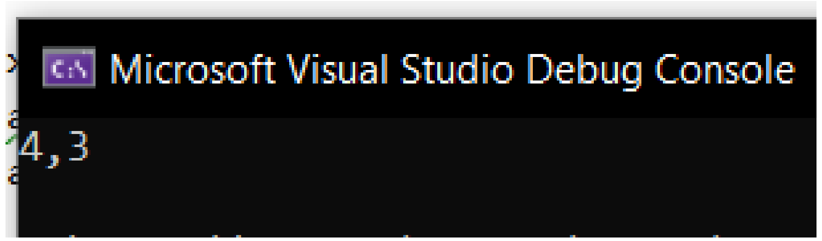
- Example

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

myclass::myclass(int a,
{
    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;
}
```

Return type is **myclass** so as to facilitate assignments and cascaded expressions

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;
}
```



```
4,3
```

Solution! = Operator Overloading

- Example

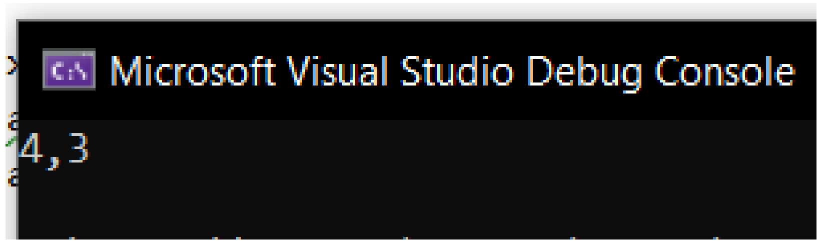
```
class myclass {
public:
    int x, y;
    myclass() {};
    myclass(int, int
};

myclass::myclass(int a,
{
    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

Return type is **myclass** so as to facilitate assignments and cascaded expressions

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
```

Operator Overloading

Two other methods which keeps “*Data Encapsulation*”:

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

Operator Overloading

Two other methods which keeps “*Data Encapsulation*”:

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

Incase of :

Member Function

```
result = foo + bar;
```

$\longrightarrow$

```
result = foo.operator+ (bar)
```

Friend Function

```
result = foo + bar;
```

$\longrightarrow$

```
result = operator+ (foo, bar)
```

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';
}
```

Operator Overloading (using Member Function)

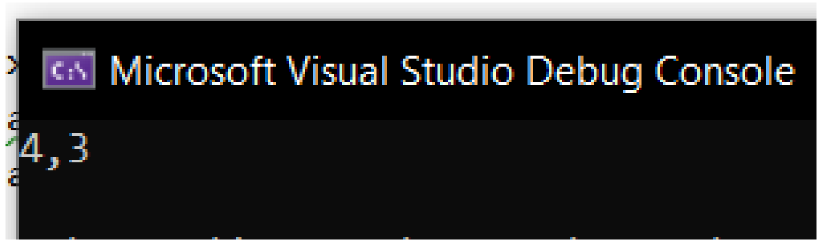
- Example

```
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;
}
```

result = foo.**operator+**(bar)



```
4,3
```

Operator Overloading (using Member Function)

- Example

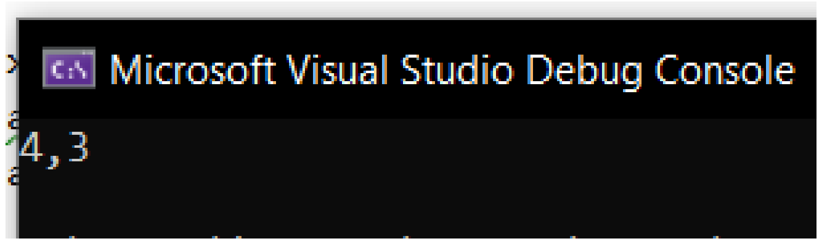
```
class myclass {
    int x, y;
public:
    myclass() {};
    myclass(int, int);
    myclass operator+ (myclass);
    void print();
};

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 = 12 + bar;
    result.print();
    return 0;
}
```

result = foo.operator+(bar)

Now what?


Microsoft Visual Studio Debug Console

```
4,3
```

Operator Overloading (using Member Function)

- Example

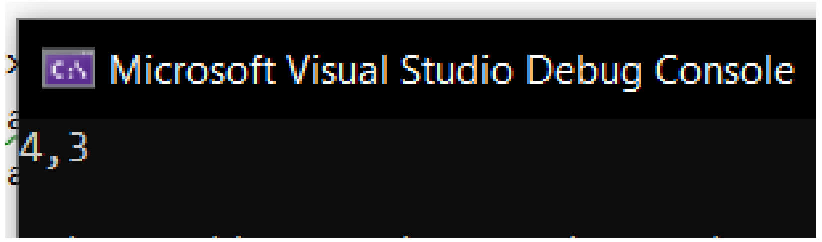
```
class myclass {
    int x, y;
public:
    myclass() {};
    myclass(int, int);
    myclass operator+ (myclass);
    void print();
};

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 = 12 + bar;
    result.print();
    return 0;
}
```

`result = foo.operator+(bar)`

Error


Microsoft Visual Studio Debug Console

```
4,3
```

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;
}
```

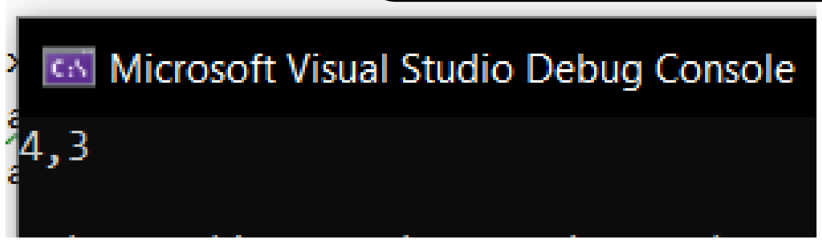
```
int main() {
    myclass foo(3, 1);
    myclass bar(1, 2);
    myclass result;
    result = 12 + bar;
    result.print();
    return 0;
}
```

`result = foo.operator+(bar)`

Error

`result = 12.operator+(bar)`

NOT POSSIBLE


Microsoft Visual Studio Debug Console

```
4,3
```

Operator Overloading (using Friend Function)

SOLUTION

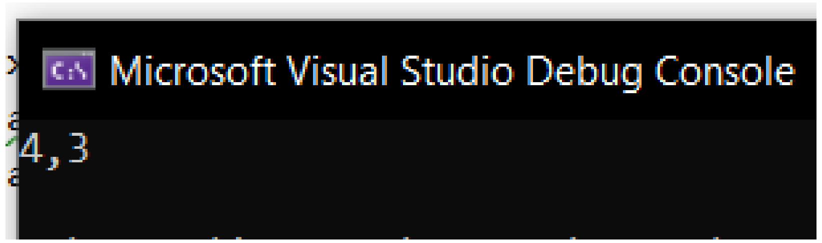
Operator Overloading (using Friend Function)

- 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;
}
```


Microsoft Visual Studio Debug Console

```
4,3
```

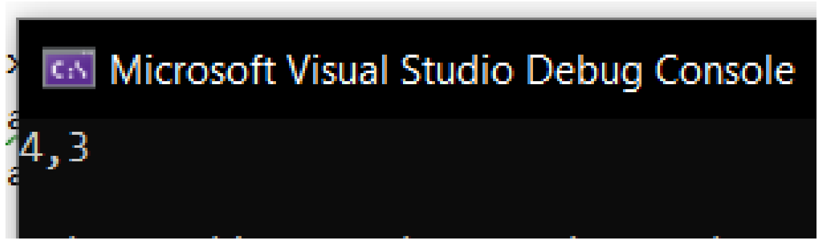
Operator Overloading (using Friend Function)

- 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;
}
```


Microsoft Visual Studio Debug Console

```
4,3
```

Operator Overloading (using Friend Function)

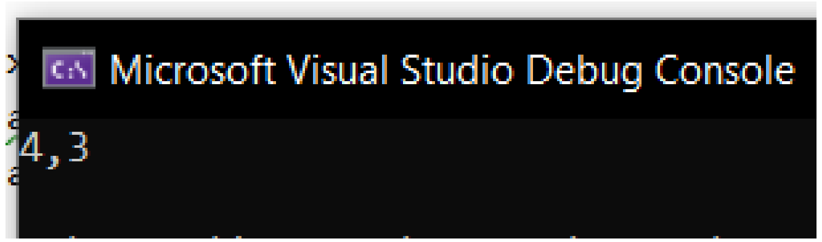
- Example

```
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;
}
```

```
result = operator+(foo,bar)
```



```
4,3
```

Operator Overloading (using Friend Function)

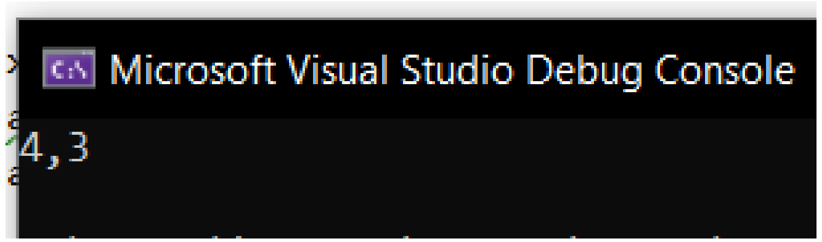
- Example

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

```
int main() {
    myclass foo(3, 1);
    myclass bar(1, 2);
    myclass result;
    result = 12 + bar;
    result.print();
    return 0;
}
```

Correct



Microsoft Visual Studio Debug Console

```
4,3
```

Operator Overloading (using Friend Function)

- Example

```cpp
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';
}
```

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

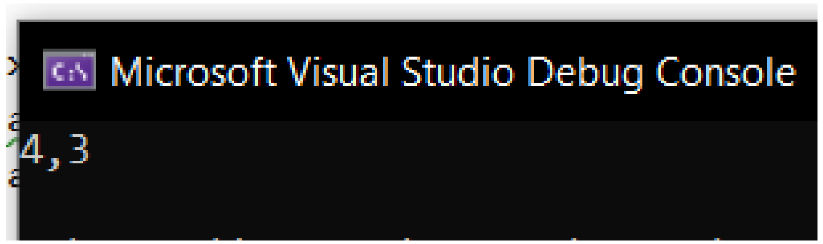
In friend function, the "**operator**" function must have at least one parameter of type class (**user defined type**)

- Following is an error:

  `int operator + (int, int);`

```cpp
int main() {
    myclass foo(3
    myclass bar(1
    myclass result;
    result = 12 + bar;
    result.print();
    return 0;
}
```

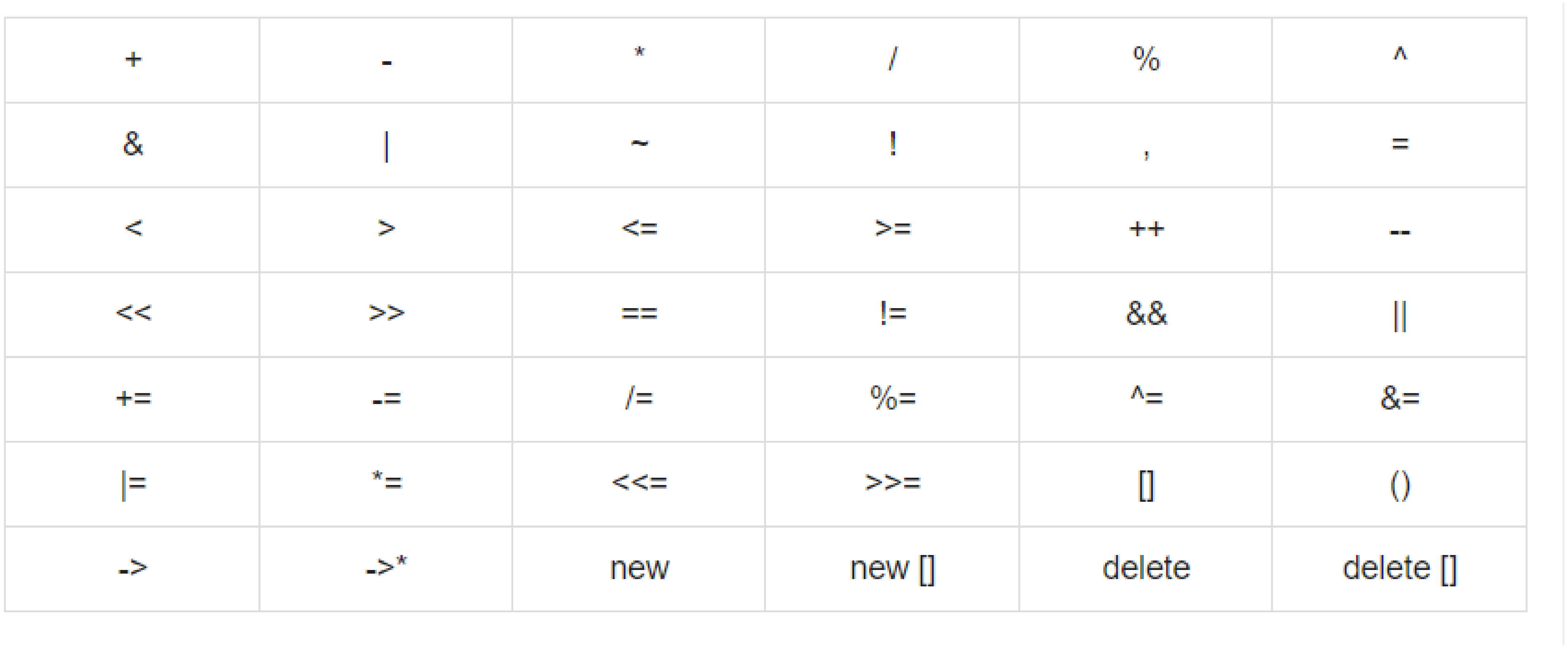
Correct


Microsoft Visual Studio Debug Console

```
4,3
```

Operator Overloading

Overloadable Operators:



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

Operator overloading

- List of operators that can't be overloaded:

```
.     .*     ::     ?:     #     ##
```

- **Reason:** They take name, rather than value in their argument except for **?:**
  - ❖ **?:** is the only ternary operator in C++ and can't be overloaded

Operator overloading

- The precedence of an operator is **NOT** affected due to overloading
- Example:
  - **`foo*bar+baz`**
  - **`baz+bar*foo`**
  - both yield the same answer

Operator overloading

- Associativity is **NOT** changed due to overloading
- Following arithmetic expression always is evaluated from left to right:

```
Example:  foo + bar + baz
```

$\longrightarrow$

Operator overloading

- Unary operators and assignment operator are right associative, e.g:
  - **`foo=bar=baz`** is same as **`foo=(bar=baz)`**
- All other operators are left associative:
  - **`foo+bar+baz`** is same as
  - **`(foo+bar)+baz`**

Operator overloading

- Always write code representing the operator
- Example:
  - Adding subtraction code inside the + operator will create chaos

Operator overloading

- Creating a new operator is a syntax error (whether unary, binary or ternary)
- You cannot create **$**

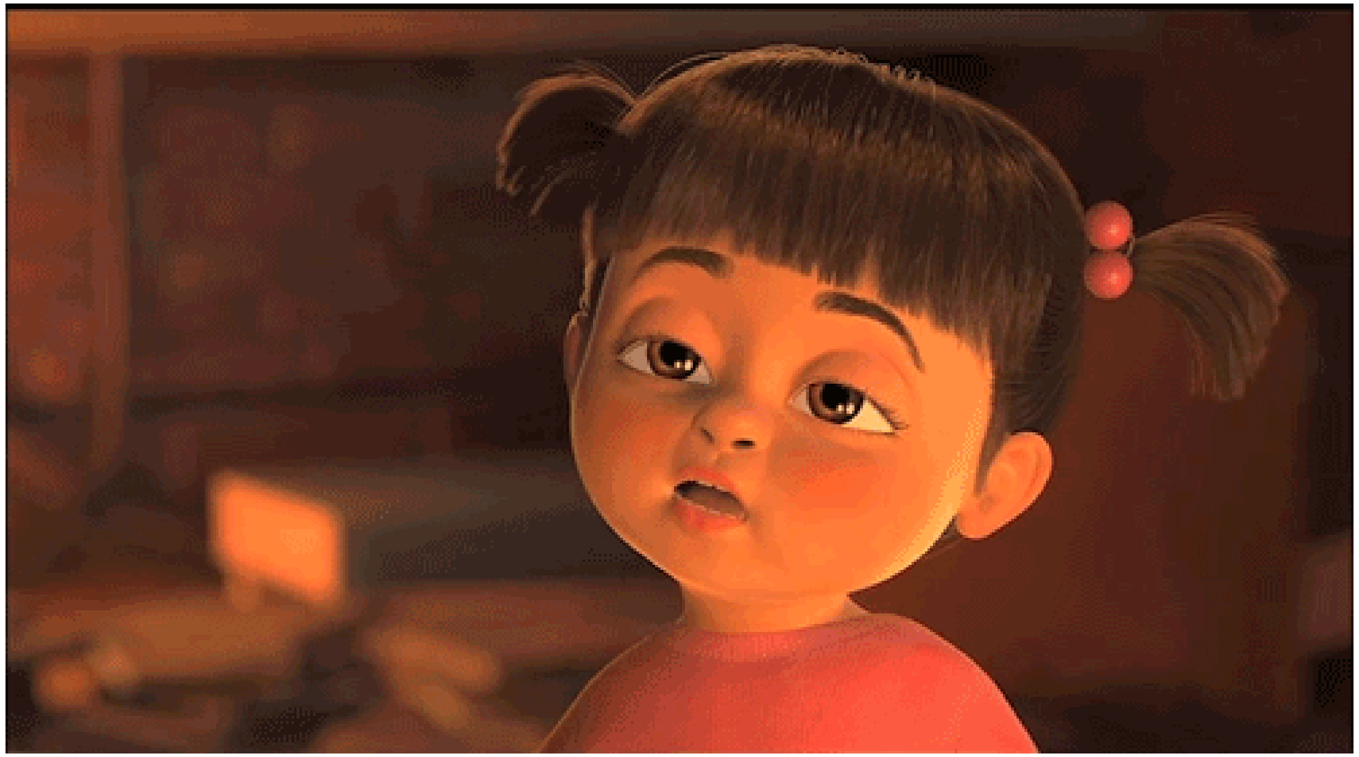
Class: Operator Overloading

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.

Thanks a lot



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