# Operator Overloading

You can’t have two functions with the same name. The compiler doesn’t know how to pick between them.

The C++ compiler is smarter. Functions can have the same name but different arguments and it can pick among them based on how you call them.

int add(int a, int b) {

return a+b;

}

int add(int a, int b, int c) {

return a+b+c;

}

void main() {

int c = add(1,2);

int d = add(1,2,3);

system("pause");

}

The return-type does NOT count since you aren’t required to use the return type when you call a function. Only the name and the arguments contribute.

These functions are said to be “overloaded”.

You can overload methods too.

The compiler will make a best guess. If you think it might guess wrong then explicitly cast:

int c = add( (int)1, (int)2 );

C++ adds “default” values.

int add(int a, int b, int c=0, int d=0, int e=0) {

return a+b+c+d+e;

}

int c = add(1,2);

int d = add(2,3,4);

Defaults go on the right and must all go together.

## Operator Overloading

class Point {

public:

int x;

int y;

Point(int nx, int ny) {

x = nx;

y = ny;

}

void scaleCoordinates(int value) {

x=x\*value;

y=y\*value;

}

};

void main() {

Point a(2,4);

Point b(3,5);

a.scaleCoordinates(25);

cout << a.x << " " << a.y << endl;

system("pause");

}

Change the scale function name to “operator\*”. Just a name.

void operator\*(int value) {

x=x\*value;

y=y\*value;

}

a.operator\*(25);

a \* 25;

When the compiler sees you using an operator on types it doesn’t know then it changes the code.

“one N two” …. Becomes … “operatorN(one,two)”

So … “a \* 25” … becomes … “operator\*(a,25)”. Does this function exist? Remember that the “member” is just a function with the hidden “this” parameter. You can move the method out of the class:

void operator\*(Point& p, int value) {

p.x=p.x\*value;

p.y=p.y\*value;

}

Same code. You just have to spell it out. AND you have to have permission.

How does the compiler handle this: “cout << p” (Point)

operator<<(cout,p).

We create this method:

void operator<<(ostream& os, Point& p) {

os << p.x << " " << p.y;

}

Point a(2,4);

Point b(3,5);

operator<<(cout,a);

cout << a;

We can’t move the function into the “ostream” class as we did with point because we can’t change the source of ostream.

Permissions might be a problem. We can solve them using a “friend”:

class Point {

friend void operator<<(ostream&, Point&);

friend class Line; // Line methods can now access Point privates

int x;

int y;

public:

};

void operator<<(ostream& os, Point& p) {

os << p.x << " " << p.y;

}

Now chain some streaming together:

cout << a << b << endl;

operator<<(operator<<(operator<<(cout,a),b),endl)

The return value from the operator<< is passed to the next call. But our function doesn’t return anything.

ostream& operator<<(ostream& os, Point& p) {

os << p.x << " " << p.y;

return os;

}

Same exercise for cin.

## Assignment, equality, copy-constructor

// int setXanYto(int c) {

void operator=(int c) {

x = c;

y = c;

}

//a.setXandYto(10);

a = 10;

operator=(Point& p) … assignment

Point a;

Point b;

a = b;

Point(Point &p) {

...

}

Copy-constructor used for copy-and-construct at same time:

Point a = b;

The operator==(Point& p) to compare:

if(a==b) {}

By default you don’t get one. Maybe you only care about an id. Maybe you need to do a string compare. This lets you control.