One of the great things about C is being able to overload some of the operators. So we can overload the standard operators like plus, minus, multiplication, and we can have them work with new objects, new classes that we have created. So let's look at how that will work. So one of the things that I end up having to do a lot is I have a vector as part of my data structures, and I want to be able to print my vector. And there isn't a standard way of printing the vector, but I would like to create an integer vector, in this case 10 to 50, and just use cout as part of the vector.

So if to be able to do that, I need to define how a vector, a vector of integers is going to get printed. So I'm going to overload the insertion operator. And when overloading an operator, we have to stick to the template because there is a lot of insertion operators. There is an insertion operator about how you print a double number, how you print possibly an object. So the operator insertion is going to take a output stream by reference, and it's going to take whatever the object as a constant reference that I would like to turn into, put into the operate output stream, and then it returns the output stream by reference.

So the way I want my vector printed is a square bracket at the beginning, a square bracket at the end, and if the vector is not empty, I would like to print the first item vector, index zero. And then once that's printed, I want to print the rest of the items using comma, next item, comma, next item, comma, next item. That way I don't have to have a if statement inside my for loop, I print one item and then it's comma, next item, comma, next item. And then finally I have the square bracket as part of it. So now continuing with our program, we have defined the operator insertion.

Now when we try to print a vector using cout, this output stream is generated and on the terminal we see 10 to 50 with square brackets. So we can also do it for any kind of object. Normally C doesn't know how you would like the object printed. So let's say we have a class A and it takes integer 10 and we'd like to print it. So here's our class A, and it's not a very exciting class.

It just has one integer as part of it. And every time we create it, we put out a line saying, hey, we have created a new A object. And once again we define the operator insertion that's going to take a output Stream by reference, constant reference of the A object and return output stream by reference. And the way I would like to print it is using a colon and then to value X as part of it. As part of that.

It's really important not to use cout as part of it. We really want to need to use the operate output stream, because this way we can use this insertion operator to turn an object into a string, to have an object be printed on the terminal using cout, or maybe printed into, inserted into a file using file operations or some other things. So output stream is much more flexible. So operator insertion takes a O stream as its input and returns an O stream as its output. So now that we have defined our A object, we are going to use the cout.

And if we go into how cout is working, we can click on here and we are in the operator insertion. So OS is getting a value, we are returning that value, and we have printed out a colon 10 as part of it. We can implement also other operations on our newly defined class A if we wanted to. So for example, we can decide that if we have two A objects, one with 10, one with 20, we already saw how we can print them. So a A one is 10, A two is 20.

We can also define how plus is going to get defined. So adding two A objects creates a new objects, new A object. In this case, it's going to have a value 10/20 30. So we get the sum 30. Or we could have it be a product.

We'll have to look at how multiplication is defined. We can also have it as a prefix increment. With a prefix increment, the object is incremented first and then the rest of the statement is executed. So now A1 is going to go from 10 to 11, and we can also have a postfix increment. In this case, the value of a 2 gets assigned to the variable x and then the value of a 2 is increased.

So a 2 which was 20 is going to become 21.

But because we have used the postfix increment, a 2's value was first assigned to x. So x's value is still going to be 20. We can also, for example, define the plus equals operator saying that hey, a 1 which is now 11 plus equals 21, which is going to modify a 1 as part of it. So a 1 becomes 32. So let's have a quick look at these operations.

And so operator plus is defined using this, the using the expected template and sorry, expected pattern, where we take two constant references to the object A1 and A2, and we create a new object A that is going to have the X values added together and we return things by value. Similarly, multiplication, we are creating a A object and returning it by value for the prefix operator. And we are taking an A by reference. And this time it's not a constant reference because we are going to modify the A object, we increment the value of x and we return A as part of it for the postfix. Well, it's kind of difficult.

How do we tell the compiler that this function is for prefix, this function is for postfix. So the way it gets done is for the postfix increment operator, there is a second argument, just int, nothing else. The second argument is int first, which indicates to the compiler that this is going to be a postfix increment operator. So now we assign a temporary value to the incoming parameter A, we increase the value of X and then we return the temp. So we return the version of the parameter A before it was incremented.

And in the case of plus equals, we are going to have two A objects. The first one is passed by reference because it's going to get modified. The second one is passed by constant reference because it doesn't need to change. And what we are we don't need to have a return value in this case. And we are setting the A1X to be plus equals A2X.

So X's value is increasing. So these are the different ways where we can implement operator overloading for C classes.