In this video, we'll look at basic input output, using cin

and cout.

Cout, cin, cerr and clog are defined in the c++ standard.

To use them, you must include iostream.

C++ uses a stream abstraction to handle IO on devices like

the console and keyboard.

Cout is an output stream that defaults to the console or the screen.

Cerr and clog

are also output streams that default to standard error and standard log, respectively.

And finally, cin is an input stream that defaults to the keyboard.

The insertion operator is used with output streams,

and the extraction operator is used with input streams.

Let's look at how we can use cout and the insertion operator.

The insertion operator inserts the value of the operand to its right.

In this case, the variable data into the cout output stream.

Since cout is by default the console,

the value of data will be displayed on the screen.

Since we're using stream abstraction,

we can chain multiple insertions in the same statement.

This makes basic IO very, very easy to do.

It's important to understand that the insertion operator does not automatically add line

breaks to move to the next line on the console.

You must do this explicitly either by using the n line manipulator

or by including a new line character, the /n you see in the last example.

If you use the end line stream manipulator, it will also flush the stream,

this is important to know since if the stream is buffered,

it may not get written to the console until it's flushed.

We'll talk more about this when we talk about file streams since it's more applicable there.

Let's see how the extraction operator works with cin.

The extraction operator extracts information from the cin input stream which

defaults to the keyboard

and stores the information into the variable to the right of the extraction operator.

The way in which the information is interpreted is based on the type of the variable.

So in this case, if data is an integer then an integer representation

will be read from the keyboard.

If data is a double then a real number will be read and stored.

If data is a string a sequence of characters will be read and stored.

Extraction operators can also be chained. In the second example,

two variables data1 and data2 will be assigned values read from the keyboard

based on their type.

The characters entered using the keyboard will only be processed when the enter key is pressed.

Cin extraction uses white space that is

spaces, tabs, new lines as terminating the value being extracted.

That's important to understand. So if you put spaces between the things that you type in,

the spaces will be ignored.

It's possible that the extraction operator could fail.

For example, suppose you want to read an integer and the user enters my name Frank.

In this case, the operation fails and the data will have an undetermined value.

Cin extraction is very easy and useful, however,

we'll later learn about other methods we can use with cin

that give us more control over the data that's being input.

Finally, we can use the same operators to insert and extract data from file streams.

This is pretty cool since the syntax is already familiar and the behavior is very similar.

We'll work with files later in this course.

Now let's head over to the CodeLite IDE and go over some examples in live code.

Okay. So now I'm back in the CodeLite IDE,

and I've got a section 5 workspace, within the workspace, I've created a basic

IO project. And this is the main file for that project.

Now what we're going to do is we're going to do some basic IO using cin and cout,

and I'll show you some of the nuances so that you really understand what's going on here.

First of all, remember, on line 4, we've got to include iostream.

And on line 6, I'm using namespace standard,

just to make the code a little bit easier to read.

Best practice would be to actually include standard

scope resolution operator cout and so forth,

as we talked about in the last lecture.

So first thing we'll talk about is this statement on line 10 right here,

cout, insertion operator, a string hello world.

We've got another insertion operator and then we've got the endline manipulator.

What this does is, it prints hello world to the console

and then advances the cursor to the next line.

Okay. I'm going to run this and I'm going to run it by pressing ctrl F5, which is

the same as run here.

So when we build and run, you can see it prints out to the screen hello world

and then it goes to the next line and this time elapsed message prints out after that.

Okay. Pretty straightforward. If I leave out the

end line

and I'll do that really quickly and I'll run it again,

you could see that it prints out hello world. It does not advance to the next line,

and then everything just prints out right after that.

Okay. So that's pretty straightforward.

Let's take a look at a couple of other examples.

Look at this example here. I'm going to uncomment this out.

First, I'm displaying hello,

and then I'm displaying world and then I'm displaying an end line at the end.

Notice that there is no end line after the first hello.

So you would expect that these two should run together with no spaces.

So we should see hello world all bunched together. So let's try that,

and you can see that that's exactly what happens.

You can see hello here followed by world with no spaces.

It does advance to the end to the next line because we have that end line manipulator here.

Okay. So let's do a few more.

And as you can see by default,

right here when we use the insertion operator with a string in this case hello,

it does not advance to the next line until -- unless we do it specifically

and explicitly. So that's what end line does.

So here are four more output statements. The first one just says cout hello world,

just like we've done before and advances the line.

The next one change the insertion operators.

So you could see that I'm putting hello on the output stream,

then I'm putting world on the output stream.

And you can see there's a space right before that w there,

and then I'm printing a new line.

So we're getting the same effect as the previous one.

Same here, I could say hello world. But here, in this case,

you can see that I'm using a .

This character right here is a new line character.

So what happens is you can embed that new line character in the string

and it will advance to the next line.

So it's similar to end line. It doesn't flush the buffer but it still advances to the next line.

And then the last line you can see, you can see that there's a couple things going on here.

You can see that I've got a new line here,

a new line here and a new line here.

So you expect us to print out hello and then go to the new line

and then print out out here

then go to the advanced to the next line and print out there

and then advance to the next line here.

So that's what. We expect we expect the first three statements

on lines 15 through 17 to print out exactly the same thing that's

the same effect and the last one to be a little bit different.

So let's try this out

and we'll compile and run here.

And you can see hello world, hello world, hello world

exactly the same and then we have hello out there one word per line.

And remember, even though that this is all in one string literal here,

we still have the s there, which -- that's how they work.

You embed them inside those strings.

Okay. So now let's do a little bit with variables.

We'll talk a lot about variables in the next section.

But I'm going to declare three variables right here, and I'll explain that to you real quick. They're really easy.

In this case, I'm declaring three variables: num1 num2 and num3.

Those are the names. Num1 is an integer, and num2 is an integer.

And then num3 is a double. That's a floating point number like 2.5 or 100.7,

a real number. We'll use num2 and num3 a little bit later.

But right now, we're only going to use num1. So I'm going to comment this line,

uncomment this line. And you can see what's happening here.

I'm saying I'm prompting the user enter an integer.

Then I'm reading an integer using the extraction operator.

I'm extracting an integer from the input stream,

and I'm storing it in num1. You can see it's going right in here.

And then I'm simply going to output. You entered whatever they typed in.

So let me explain how this works. Here's your keyboard.

And remember, cin is by default tied to your keyboard.

We can redirect it as we wish but by default it's the keyboard.

Now what happens is characters are read from your keyboard,

and they're not read directly by cin, they're stored in a buffer.

This is done to make the program more efficient. So if. I type in

123 on the keyboard,

then what happens is the numbers are read from the buffer,

the characters in this case are read from the buffer.

And the compiler knows that num1

is an integer.

So it's going to read those characters,

and it's only going to process what makes sense to be an integer.

So in this case, it's going to read the one and say yeah. It's going to read the two,

yep. Going to read the three, yep. Now we're done.

So we've got 123.

So 123 gets stored into num1,

and that will print out here.

Okay. Now you need to press enter when you do this on the keyboard.

And also white space is ignored.

So if I type in like 10 whites, 10 spaces

or tabs or anything that's considered white space

and then I type in 123,

then I type in a whole bunch more white space, it'll all be ignored.

Okay. So let's take a look at that in action, and I'll just clear this real quick.

And let's run this,

and it's going to say enter an integer. So I'm going to type in 123,

press enter. Remember, I have to press enter,

and it says you entered 123. Just what we'd expect.

I'll run it one more time. And this time, I'm going to type in a bunch of spaces,

I'm pressing the space bar now I'm pressing tab.

Then I'll press 123. And I'll type in some more spaces.

Now when I press enter, you can see you still entered 123.

All that beginning white space is ignored.

And everything after the 3 is considered not an integer. So this is where

if you read from cin again, it's going to start right after that 3.

Okay so that gives you an idea of what's going on there and let me

comment thatout again and we'll keep going through some more examples.

In this case, we're going to read two numbers.

We're going to say enter a first integer,

and we're going to read that into num1.

Enter a second integer, and we're going to read that into num2.

And then we're just simply going to say you entered num1 and num2.

Remember, in this case here num1

and num2, right here,

will be replaced by whatever values the user typed in. Okay.

Okay. So let's do this. Let's run this one,

and you'll see it's exactly what you would expect.

And let's see, oops I should have commented these guys out.

Let me uncomment them. And let's try that again.

So enter a first integer. Let's say we want to type in 100.

And I'm going to press enter, enter a second integer 200,

I'll press enter again,

you entered 100 and 200. Again, just what we expected.

Now let me show you something that you might not expect. Let me run this one more time,

and I'm going to enter 100, followed by a space,

followed by 200, and press enter.

You can see my output statement worked.

It says you entered 100 and 200, but the output looks a little different, right.

Well, you can see what's happening, I've entered two integers.

So what happens is when I read those integers

from the standard input, right, via the extraction operator,

it's going to grab both of them. It's not going to wait because it's got both of them in there already.

Again it's not going to wait for me to press enter two times

because it's got them in there. Let me show you what that looks like.

So you can better understand it. All right. So let's say that we're here,

and we just ran this program, and we entered 100

followed by a space, and I'll just use that character there for a space

followed by 200.

Okay so right here it's waiting for me what I did was I entered 100

space 200 and I pressed enter.

The extraction operator realizes that num1 is an integer,

so it's going to try to read an integer from this stream here.

It's going to start right here and it's going to say I got a 1 a 0 and a 0.

Boom I hit a white space that must mean that my integer is done.

So it's going to take 100 and store it right into num1.

Now it's going to output, enter another integer.

And it's going to try to read that second integer from the input stream.

Well, it's already in there, right. So it's going to start right here, and it's going to say

200, hit the end, aha that's 200.

It's not going to wait for me to press enter because there's already data

in that buffer. So in this case, the 200

gets put into num2 and this will print out 100 and 200.

So your output will be correct

but the way that the the prompts are displayed is a little

different because it's got data in that buffer already.

Same thing with -- let me run this one more time and I'll show you a little bit

of a different example.

Same thing with space 100 space

tab 200 or 200.

Yeah, you can see you entered 100 and 200, it's just ignoring all that space,

but it's got those two values in that stream. It's just going to process the stream.

It doesn't wait for you to press enter twice.

Okay. So I'll comment those out, and

we can chain the extraction operator,

just like we we can chain the insertion operator.

So in this case, you can see that we've got the cin

followed by an extraction operator right there,

followed by another extraction operator, right.

So what I'm doing here is the the output statement for the user is a little different.

It just says enter two integers separated with a space.

So let's say I enter 100 a space and then 200.

Now this guy is going to read two integers from the input stream.

And it'll assign 100 here and 200 here.

Okay. Let's do that.

And you can see right here enter two integers separated with a space.

Remember, I could put leading spaces too if I want, but in this case I'll just say 100

space 200.

And it says here you entered 100 and 200.

This is actually a really handy way to get information from the user when you want them to

you know enter more than one number,

enter three numbers separated by spaces. That's really easy for them to do

rather than say enter number one, enter number two, enter number three and so forth.

Okay. So that's another option you've got.

Okay. So I'll comment this out. And now we'll process some doubles.

In this case, if you remember, let me scroll up just a little bit, if you remember

num3 was a double, right.

So it's a floating point number, a real number.

And what I'll do here is I'll just uncomment this out.

And the prompt to the user is enter a double

right here.

And you're going to read the double into num3 from the input stream.

Again, here's the input stream.

Suppose I typed in 12.5.

That's a real number.

It realizes that it needs to read a real number from the input stream

because num3 was defined up top as a double.

So it's going to come here. It's going to read a 1, it's going to get the 2,

it's going to see the decimal point that's valid in a real number. Then it's going to read the 5.

Then it's done. And 12.5 will be assigned here

and will be displayed down here.

Okay. So let's run that.

Enter my double, I'll press 12.5 and enter.

And it says right there, you entered 12.5.

Just like we expected.

Okay. So let's do this last one.

And I'll show you what could go wrong here.

There's always something that could go wrong, right. So let me uncomment that.

And look at this example here. This one is saying

enter an integer,

and you're going to read the integer into num1.

And then it's saying enter a double, and you're going to read the the double into num3.

Okay. Now if we run this and I type in, let's say 10,

and 12 space 12.5,

that's not a problem at all, right, because it's going to read the integer

and it's going to say 10.

I see a space. I'm done. The 10 goes into num1.

Then it's going to read a double here, num3.

It's going to say 12 decimal point 5, 12.5,

perfect. It's going to go in here.

The integer is 10.

The double is 12.5, and that'll work just fine.

Let me run that just so you can see that.

And then we'll do one more example -- a couple more examples where you can see some of the issues here.

So there's my integer. Let's say, it's 10

and the double is 12.5.

And you can see the integer is 10 and the double is 12.5.

Okay. But what happens if I enter something like,

let's say, 10.5

something like that.

Think about this logically, you're probably guessing what's going to happen.

It's going to read an integer. So it's going to see that

the 1 and the 0 and it's going to say yep that's an integer. When it sees the decimal point,

it knows it can't be an integer anymore. So it's just going to take the 10

and store it here.

Then you're going to say read me a double.

Well, it's going to start at the decimal point, and it's going to read 0.5.

0.5 is a double, right. So it's going to store 0.5 here.

So your output is going to be 10 and 0.5.,

maybe not what you expected but you can see the behavior makes perfect sense. You're telling it

what to do. You typed in 10.5. Well, 10 is an integer and 0.5 is a double and

I can read those.

So let's try that with 10.5., and we'll see what that looks like.

So let's enter the integer 10.5. Now remember,

there's going to be that 0.5 left in the buffer. So it's not going to ask me.

It's going to ask me to enter the double,

but it's not going to wait for me to press enter because it's already got data in the buffer.

So I press enter and you can see

enter a double, right, it's not waiting for me here.

The integer is 10, and the double is 0.5.

Now the last example is, let's run this one more time and we'll do something really weird here,

we'll just type Frank.

Now that's really wonky because you've just told it that

you're going to read an integer and a double and you just gave it a string of

characters here. So if I press enter,

you can see what's going on here. The integer is zero,

and the double is some undefined crazy small number here.

Okay. Well, what happens is when it tried to read the integer,

it got the F in Frank and it's lost.

So the input failed, and it goes into a fail state.

From that point on,

everything that's happening is unreliable. You can see that the integer got 0.

It got basically initialized to 0 there. But everything else

after that point is completely unreliable, you've got to be really careful.

Now in real-world, programs

we're not really using cin to to read characters.

We're using strings. We're reading them into strings, and then we're trying to convert those strings

to numbers or whatever else we really need.

But they're really, really handy and easy to use, and that's what we're going to do for the

vast majority of this course.

We're going to use cin and cout so that we can read numbers,

and we can output numbers or strings or whatever we like.

So that finishes off this video.

This is basic io using cin and cout. You can see it's pretty straightforward.

Don't worry about mistakes right now as we're working through these programs.

If it says enter an integer, enter the integer. If it says enter a string, enter the string.

Don't worry about error handling that comes later.