Lecture 4 — Console Input/Output; Comments

J. Zarnett
jzarnett@uwaterloo.ca

Department of Electrical and Computer Engineering
University of Waterloo

August 27, 2016

Acknowledgments: D.W. Harder, W.D. Bishop, N. Armstrong

Part I

Console Input/Output

Garbage In, Garbage Out

— Common Programmer Saying

Our First C++ Program

```
#include <iostream>

using namespace std;

int main()
{
    cout << "Hello World!" << endl;
    return 0;
}
```

Console Output

In the Hello World program, we already had our first introduction to console output: `cout << "Hello World!" << endl;`

Then later, we showed console output of the `'X'` character.

Let's take some time to examine this idea of console output.

Console Output

We can output text (enclosed in the " double-quotes character) to the console, just as we can print numbers or variables.

```
int numberOfPages = 99;
cout << numberOfPages << endl;
```

When using `endl` ("end line"), at the end of printing, the program output goes to the next line.

If you do not want this behaviour, simply do not include the `endl`.

In this way, you can print multiple things on the same line.

Output Example

```
int numberOfPages = 99;
cout << "The Number of Pages is" ;
cout << numberOfPages << endl;
```

There is a problem with this when printed to the screen.

[Demo: In-class examination of what's wrong here]

Strings

So far we have seen a few examples of text shown in the double quotation marks.

This is another kind of type, called the `string`.

Examples we have seen so far:

```
"Hello World"
"The Number of Pages is"
```

We could also have written:

```
string helloWorld = "Hello World";
cout << helloWorld << endl;
```

Strings

The length of a `string` may be zero or more characters...

The `string` resembles the `char` in many ways.
We'll examine this subject in more detail later on.

For now, however, just think of a `string` as a grouping of characters.

New Lines in Output

We can go to the next line by using `endl`, but there is another way to put a new line wherever it is needed.

There is a special character that indicates to the console to go to the next line, but that character is not found on the keyboard.

To signal that character, we need to use an escape sequence.

Escape Sequence

An escape sequence already appeared in the slides when we looked at `char` variables. `char example = '\x0058';` includes one.

A backslash (\) preceding a sequence of characters tells the compiler that the characters following the backslash have a special meaning.

There should be no space between the backslash and the character to which it applies.

To put a backslash in text to be output, you must escape the backslash: (\\).

Escape Sequence Examples

Escape Sequence	Meaning
`\n`	New line
`\t`	Horizontal tab
`\'`	Single quote character
`\"`	Double quote character
`\\`	Backslash
`\xAAAA`	Hex number AAAA
`\uXXXX`	Unicode character XXXX (XXXX in hex)

This is not an exhaustive list.

Don't confuse the backslash (\) with the forward slash (/).

Outputting a New Line Character

The escape sequence for a new line is `\n`.

Thus the following two statements produce the same output:

```
cout << "ABCD" << endl;
cout << "ABCD\n";
```

Any escape sequence like `\n` must be inside the string literal, or in a char variable.

When displaying something that isn't already a string, it is converted to a string before being put out to the console.

Later in the course, we'll see exactly how this happens.

Console Input

Now let's examine console input (reading data into the program).

Console input can be performed using the `cin` command.

This command is fairly clever and will convert the user input to an appropriate type, if it can be done.

```
int a;
cin >> a;
```

This takes user input at the console, converts it to an integer, and stores it in variable a.

Note: the angle brackets go the opposite way from a `cout` command.

Console Input

It is possible to read in multiple values in one line.

Example: `cin >> a >> b;`

This is equivalent to:

```
cin >> a;
cin >> b;
```

In console input, a space, tab, or new line separates the two values.

It is also possible to read in a `string` at the console:

```
string word;
cin >> word;
```

Suppose we wanted a `string` to be two words, like the "Hello World" example from earlier. Can this be done?

Reading a Line

Normally a space ends the input, so typing in both words with a space will not work. We could read the entire line of input as one `string`.

```
int main()
{
    string text;
    getline(cin, text);
    return 0;
}
```

Here, `getline` is a function (a concept we will return to soon).

Its purpose is to take a line of input from its source (`cin`) to its destination (variable `text`).

Reading Input and Execution

When you have a statement with `cin` or `getline` in your program, when the execution gets to that statement, what happens?

Answer: the program waits for the user.

Later statements do not execute until the read statement is complete (e.g, the user presses enter and you store his/her input in a variable).

It is a good idea to indicate to the user you are waiting for input. Otherwise, the user may not realize the program is awaiting input.

Reading in Numbers

Suppose we asked the user for a number rather than some text.

The reading options operate on strings (or characters) – not numbers – and yet somehow the data was stored as an integer.

There is a process to convert text from a `string` to a numeric type.

The process of conversion from a string is called parsing.

More formally: parsing is the act of extracting data in a particular format from a string of text.

Parsing Numbers

Analogy: writing down a number someone says out loud.

When they speak, they say “eight six seven five three oh nine”

You listen to the spoken words and convert their sound representation into a written representation: 8675309.

Parsing Numbers

To parse a string, it's important to know what kind of number it is.

Trying to read a number like 456.149 into an `int` may be a problem.

The syntax for parsing is `atoi` for integers, and `atof` for floating point numbers.

```
string costString = "9.99";
double costNumeric = atof( costString );
```

Parsing Numbers

Of course, you would rarely want to convert a literal like `"9.99"`.

Much more commonly, you convert something the user typed in.

If the user gives in some invalid input, we might see an error.
Later on in the course, we'll discuss how to handle errors.

The next slide shows an example program combining all the things we've discussed in this lecture so far:

Combining Parsing, Input, & Output

```
#include <iostream>

using namespace std;

int main()
{
    string s;
    double d;

    cin >> s;
    d = atof( s );
    cout << "d = " << d << endl;

    return 0;
}
```

[Demo: running this program to see what happens for (in)valid input]

Part II

Comments

Code Comments

A comment is some text in the source code of a program that is ignored by the compiler.

It is intended to be significant to other programmers (or oneself) when reading the code later.

Good comments make the code easier to understand.

Code Comments in C++

The recommended way to add a comment is to use two slashes: `//`.

All the text between the `//` and the end of the line will be considered a comment and is ignored by the compiler.

```
force = mass * acceleration; // Compute the force
```

The text “Compute the force” appears after the `//` so it is a comment.

(There is a less common way that uses `/* */`, but this is a legacy from C and is not recommended).

Code Comments

Comments can appear above code. The double slash may start a line:

```
// Compute the force
force = mass * acceleration;
```

When programming, you might use comments to write out the steps in plain language before you write the actual code.

If you want to temporarily remove a statement from your program, you can put the // in front of it to "comment it out".

Then when you want to add it back, just remove the // and you don't have to rewrite the statement.

Code Comments

The lecture notes may contain lines that are commented to explain what that line or group of statements do/does.

It is not necessary to put a comment on everything.

Because they are ignored by the compiler, comments have no effect on the code's execution or performance.

Good comments help others understand what code does and why.

Without Comments

It's unfortunately easy to write code that is difficult to update/change.

If you do not document code, you can find out exactly what it does, but you have no understanding of why it is done that way.

To add new functionality, we must determine:

1. What the code does
2. What the problems are that the programmer is trying to solve
3. How the problem is being solved
4. Why the given approach was taken

Without Comments

If there is only the code, the reader must figure the four items on the previous slide out for him/herself.

This can lead to many wasted hours. Code is often written once but read many times.

If you as an engineer do not comment code, you will not expect this of your fellow programmers.

Don’t write good comments? Maintenance costs skyrocket.

If maintenance costs skyrocket, it’s bad for the company & your career.

Writing Good Comments

Especially with new coders, writing this seems like a good comment:

```
// Add 4 to x
x += 4;
```

The other developers would rather see:

```
//Pad x so that the label doesn't run into the textfield
x += 4;
```

Comment about why, not what.

Choosing good identifier names can reduce the need for comments.