CS 11 Data Structures and Algorithms

Lesson 1: C++ Basics

Skip to Main Content

Section 1.1: Intro to Output and Program Structure

A computer program is a sequence of instructions that tell the computer, step by step and very carefully, what it is we want it to do. (There are other ways to view computer programs which we will discuss later. For now this is the best way to think of a computer program.) For example, if we want the computer to bake a cake, we might include in our program instructions like "mix together the flour and sugar", "preheat the oven to 350°", and "bake for one hour". The instructions in a computer program are called **statements**.

The first thing we ought to be able to tell our computer to do is to write something on the screen. C++ uses the following statement to instruct the computer to write the words "Hello world!" on the screen.

```
cout << "Hello world!";
```

When we want to tell the computer to write something on the screen, we start by saying `cout <<`. The word `cout` is used to repesent the computer screen. In technical C++ language it is called a **stream**, meaning a stream of characters. The symbol "<<" is called the **stream insertion operator** because it is used when we want to insert something into a stream. The thing on the right of the stream insertion operator in this case is a **string**. A string is a series of characters surrounded by quotation marks. The arrows point left to indicate that we are taking the thing on the right (a string, in this case) and inserting it into the thing on the left (a stream that represents the computer screen, in this case).

Like all statements in C++, this statement ends with a semi-colon.

We can't, however, just give the computer a statement like this all by itself. Before the computer can execute a statement, the statement has to be in a program. Here's a complete C++ program, along with a picture of what appears on the screen when we run it (we will call what appears on the screen **screen output**).

```
#include <iostream>
using namespace std;

int main()
{
    cout << "Hello world!";
}
```

```
Hello world!
```

Don't let all the strange stuff in this program scare you! For now all you need to know is this: whenever you want to write a C++ program, you need to copy everything exactly the way it looks in our example, except that you must remove the `cout << "Hello world!";` line and insert your own statements in its place. The following box explains some parts of the program. For even more detailed descriptions of the various parts of a C++ program, you should consult your text. You do not, however, need to understand these details to write a C++ program. They are given here only to satisfy your curiosity. You may skip them for now, if you want to.

- The line `#include <iostream>` tells C++ to include in your program the code found in a file named "iostream". This is the file where the word "cout" is defined, so you must have this line in any

program that uses the word "cout". Every program you write this semester will use the word "cout", so you'll have to include it in all of your programs. It must go at the very top of your program.

- The line `using namespace std;` has to do with the concept of namespaces, which is beyond the scope of this course. If you were to look in the file "iostream" (I'm not suggesting that you do) you will find that when the word "cout" is defined there, it is defined to be part of a namespace called "std". So you can't use the word "cout" unless you are using the namespace "std".

- The line `int main()` says to the computer, "This is where to start executing statements." We will see later that `main()` is not always the first thing to appear in C++ programs. The computer begins executing statements where it finds `main()`, not necessarily at the top of the program.

- The open and close curly braces, { and }, mark the beginning and end of the "main" part of a C++ program. You could think of it as saying "begin" and "end".

Section 1.2: endl

All right, let's try adding a bit of complexity to our program. We'll start by writing a program which will instruct the computer to write 2 lines of text on the screen instead of just one. To be precise, let's have it write this on the screen:

```
Hello world!
It's sure cold out here!
```

Here's our first try, along with the screen output:

```
#include <iostream>
using namespace std;

int main()
{
    cout << "Hello world!";
    cout << "It's sure cold out here!";
}
```

```
Hello world!It's sure cold out here!
```

Did our program do exactly what we had intended it to do? No! We wanted our screen output to go onto two lines, but it came out smashed together on one line! Here's the problem. When we say

```
cout << "Hello world!";
```

the computer doesn't automatically go down to the next line to get ready to print whatever comes next. It stays right where it is, at the `!` in "`Hello world!`", and whatever comes next gets printed right there, as you saw in our example. So, every time we want the computer to go to the next line we need to specifically tell it to do so. We do this by adding the word `endl` (which means "end line") to our `cout` statement. Putting `endl` in a cout statement is like telling the computer to hit the return key. Here's an example to illustrate this:

```
#include <iostream>
using namespace std;

int main()
{
    cout << "Hello world!" << endl;
    cout << "It's sure cold out here!" << endl;
}
```

```
Hello world!
It's sure cold out here!
```

Now our program is working as we intended it to work. As a general rule, unless you have a reason to omit it, you should end each of your `cout` statements with a `<< endl`.

Now we know enough to discuss the `cout` statement more precisely. The way `cout` works is that it accepts a list of things that you want it to write on the screen. Each item in this list is preceded by a `<<` operator. So the individual parts of the first cout statement in the example above could be translated like this:

C++	English
`cout`	write the following list of items on the screen
`<<`	here comes the first item
`"Hello world!"`	[the first item -- a string]
`<<`	here comes the next item
`endl`	[another item -- telling the computer to hit the return key]
`;`	end of statement

It is possible for us to write more than one line on the screen using just one `cout` statement. To do this we take advantage of `endl`, like this:

```
#include <iostream>
using namespace std;

int main()
{
    cout << "Hello world!" << endl << "It's ... here!" << endl;
}
```

```
Hello world!
It's sure cold out here!
```

Notice that I had to use "..." in the code above instead of putting the actual words there. This is not a programming technique, I just did it because if I put the whole sentence there, the code would probably go off the right edge of your browser, and you might also have trouble printing it out. A better technique would be to continue the statement on a second line, as illustrated below. In this case I still have only one cout statement, but it goes onto two lines.

```
#include <iostream>
using namespace std;

int main()
{
    cout << "Hello world!" << endl
         << "It's sure cold out here!" << endl;
}
```

```
Hello world!
It's sure cold out here!
```

Placing endl in the middle of a cout statement causes the computer to go to the next line just as it does when you place endl at the end of a cout statement. However, we usually avoid this technique because it often makes programs more difficult to read.

Section 1.3: Backslash (the Escape Character)

There's one more little problem that might come up. Let's say we want a program to produce the following screen output:

```
Hello world!
Bob said, "You should wear a jacket!"
```

Here's our first try:

```
#include <iostream>
using namespace std;

int main()
{
    cout << "Hello world!" << endl;
    cout << "Bob said, "you should wear a jacket!"" << endl;
}
```

```
syntax error!!!
```

Oh no! What went wrong? Well, let's pretend that we are the computer for a minute and analyze this program. Before the computer executes anything, it reads through the program to make sure that everything makes sense. Everything seems to be fine up to the second `cout` statement. In the second `cout` statement, we have `cout`, and then our `<<` which tells us that an item is coming up. Next we see the beginning quotation mark which tells us that the item coming up is a string. We immediately look for the next quotation mark, which will mark the end of our string. We find it right after the comma, and so what we've seen so far on this line is

```
cout << "Bob said, "
```

which is fine. Now what are we expecting to see next? Well, a semi-colon would be fine, since that would tell us that we have reached the end of the statement. Or, a << would be fine, since that would tell us that another item to be written on the screen is coming up. However, we see neither of these. We see the letter "y". So we stop and say "What is a 'y' doing here!?" (or, in computereze, "Syntax error!!!")

The problem is that we wanted the quotation mark to be printed out, but C++ thought we meant it to mark the end of our string. What we need is some way to tell C++ that the quotation mark coming up is intended to be written, not as the signal that our string has ended. The way we say this in C++ is by placing a backslash (\) right before the quotation mark that we intend to be written out. When it is used in this way the backslash character is called the **escape character**. It works like this:

```
#include <iostream>
using namespace std;

int main()
{
    cout << "Hello world!" << endl;
    cout << "Bob said, \"you should wear a jacket!\"" << endl;
}
```

```
Hello world!
Bob said, "You should wear a jacket!"
```

Section 1.4: Int Variables and Values -- Declaration and Input

Let's work on a new problem now. Let's say we want a computer program to calculate paychecks for our employees. We want it to ask how many hours the employee worked and then we want it to calculate the amount of the paycheck and print out this amount. We will assume for now that all of our employees make $12 an hour. Here's an example of what will happen when we run our program (the text in bold face is typed by the person running the program. We call this person the **user**):

```
Enter hours worked: 23
The amount of the paycheck is $276
```

Let's think through our strategy for writing the program before we start writing any C++ statements. We ask ourselves, "What are the steps involved in accomplishing the task at hand?" As is the case with many programs, this program involves the following steps:

1. get the information from the user (the hours worked) and store it.
2. do a calculation (the amount of the paycheck should equal the hours worked times 12).
3. write the results on the screen.

We will now examine these three steps, step one in this section (section 1.4) and steps 2 and 3 in the next section (section 1.5).

Step 1: First, we need to have someplace to store the hours worked so that we can then multiply it by 12. Just like in algebra, we do this with a **variable**. A variable is someplace where we can store a value while our program is running. We'll want a variable called "hoursWorked" in which to store the value that the user types in for hours worked (23 in our example). In order to do this we need two new C++ statements: one that says "create a variable named hoursWorked that can store an integer value", and one that says "wait for the user to type in a value and put the value that the user types into the variable hoursWorked." Here are the statements:

```
int hours;     // create a variable named "hoursWorked" that
               // can be used to store an integer

cin >> hours;  // wait for the user to type in a value and put
               // that value into the variable "hoursWorked"
```

Here's step #1 of our paycheck program:

```
#include <iostream>
using namespace std;

int main()
{
    int hoursWorked;

    cout << "enter hours worked: ";
    cin >> hoursWorked;
}
```

Further details about this example:

- Statements like `int hoursWorked;` that create variables are called *declaration statements*. We don't normally speak of "creating" variables, we speak of "declaring" them. Declaration statements always go before any other statements in your program. (This is not technically true, but a good rule to follow for now.) **You must always declare every variable in your program with a declaration statement!**

- In this example we are assuming that the hours worked will always be an integer. That's why we put the word `int` at the beginning of our declaration statement. Later we will discuss other types of variables that can store different types of values.

- Every `cin` statement should have a `cout` statement right before it, telling the user what it is that should be typed. This `cout` statement is called a *prompt* because it is prompting the user to type something. If you have a `cin` statement without a prompt, the computer just stops and waits and the user doesn't know what to do!

- Notice that in a `cin` statement the >> points in the opposite direction from the << in a `cout` statement. The word `cin` is a stream that represents the keyboard. The symbol ">>" is called the *stream extraction operator*. The arrows point to the right to indicate that we are extracting something from the

thing on the left (a stream representing the keyboard, in this case) and putting it into the variable on the right.

- It is important to use descriptive names for variables. Notice that we chose to call our variable "hoursWorked." We could have called it simply "hours" or even "x" or "y" and our program would have worked correctly, but in order to make our program more clear and easy to understand we call it "hoursWorked."

- **Rules for variable names:** Variable names may be of any length and must be made up of letters, digits, and underscores (_). A variable name must not start with a digit. Notice that a variable name is not allowed to have a blank space in it. This means that variable names must be one word, not two.

  C++ variable names are case sensitive. This means that "hours" and "Hours" are two different variables.

Section 1.5: Int Variables and Values -- Assignment and Output

step 2:
Now we need a C++ statement that says "multiply hoursWorked times 12 and put the answer in a new variable called paycheckAmount." Here it is:

```
paycheckAmount = hoursWorked * 12;
```

We have to be very careful about how we view this kind of statement. This statement is called an *assignment statement* because its job is to evaluate the expression on the right side of the equal sign (`hoursWorked * 12`) and *assign* that result to the variable `paycheckAmount`. This is **not** an equation like you might see in algebra. It is **not** a statement of equality. In other words, we **are not** telling the computer a fact: "paycheckAmount is equal to hoursWorked times 12". We **are** telling the computer to do a very specific task: "take the expression on the right side of the equal sign, evaluate it, and put the result into the variable on the left side of the equal sign." For this reason it is dangerous to read this statement "paycheckAmount **equals** hoursWorked times 12," although many people do. It is better to read it "paycheckAmount **gets** hoursWorked times 12." In fact, this operator is not called an equal sign, it is called the *assignment operator*.

Of course, paycheckAmount is a variable, so we have to declare it before we can use it.

Step 3:
Now we just have to print out the results of our calculation. We can do this with our `cout` statement from the last section. Remember that a `cout` statement accepts a list of items to be written on the screen. The only new thing you need to know is that any of the items in a `cout` statement can be variables (instead of strings or endl's). So, putting all of our steps together, here's our program:

```
#include <iostream>
using namespace std;

int main()
{
    int hoursWorked;
    int paycheckAmount;

    cout << "Enter hours worked: ";
    cin >> hoursWorked;
    paycheckAmount = hoursWorked * 12;
    cout << "The amount of the paycheck is $"
         << paycheckAmount << endl;
}
```

```
Enter hours worked: 23
The amount of the paycheck is $276
```

String Constants Versus Variable Names:
When the program above is run it produces the output we asked for in the problem statement. Here's a

slightly changed program that doesn't work as intended. See if you can figure out what has been changed.

```
#include <iostream>
using namespace std;

int main()
{
    int hoursWorked;
    int paycheckAmount;

    cout << "Enter hours worked: ";
    cin >> hoursWorked;
    paycheckAmount = hoursWorked * 12;
    cout << "The amount of the paycheck is $"
         << "paycheckAmount" << endl;
}
```

```
Enter hours worked: 23
The amount of the paycheck is $paycheckAmount
```

Why did our program write the word "paycheckAmount" on the screen this time when what we wanted it to write was the value of the variable "paycheckAmount"?? Since we put quotation marks around the word "paycheckAmount" in our program, C++ treats it as a string, and writes it on the screen. In order to be treated as a variable, the word "paycheckAmount" must appear with no quotation marks around it. Adding quotation marks at the wrong times and leaving them off at the wrong times are very common mistakes for novice programmers, so let's spell out the difference very clearly:

```
If you want the word to be a variable, do not use quotation marks.
If you want the word to be a string, use quotation marks.
```

Using a Variable for the Pay Rate:
Here's an example of a program that extends the concepts of this section a bit further. To make our paycheck program more flexible let's allow the user to enter the pay rate instead of forcing it to always be $12. That means we'll need another variable to store the pay rate. Instead of multiplying the hours worked by $12, we'll multiply the hours worked by the pay rate. Here's what it looks like.

```
#include <iostream>
using namespace std;

int main()
{
    int hoursWorked;
    int paycheckAmount;
    int payrate;

    cout << "Enter hours worked: ";
    cin >> hoursWorked;
    cout << "Enter rate of pay: ";
    cin >> payrate;
    paycheckAmount = hoursWorked * payrate;
    cout << "The amount of the paycheck is "
         << paycheckAmount << " dollars." << endl;
}
```

```
Enter hours worked: 23
Enter rate of pay: 10
The amount of the paycheck is 230 dollars.
```

Notice that we also changed our `cout` statement so that we need to write the word "dollars" on the screen **after** we write the value of the variable "paycheck". This is to illustrate a `cout` statement that is slightly more complex.

Summary of C++ Statements:
We have discussed four different C++ statements so far. You should make sure that you understand each one. They are

- cout statement
- cin statement
- declaration statement
- assignment statement