Lecture 1 Notes

Statements

A computer program is composed of a sequence of statements. Each statement gets executed one after another. A statement may create data, read data, write data, perform a calculation, or transfer execution to another part of the program. In the C++ programming language, there are five categories of statements: declarations, expressions, selections, jumps, and iterations.

A declaration is a statement that creates a variable. Variables store data and can be referenced by name. There are different types of variables as well as different types of data. We will cover these shortly. For now, take note that C++ operates with "values". A value is an instance of data in a particular data format. Examples of values include the integer value 4, the number 9.81, the letter 'A', the text value "hello".

An expression is a statement that creates a value, whether it be the spontaneous creation of a value or the result of a calculation. Creating values, adding values, multiplying values, comparing values, and assigning values to variables are all examples of expressions.

Data types

In C++, every value contains data that conforms to a particular data type. The data type of a value must be known in order to know how the data can be interpreted and manipulated. C++ supports many data types as well as the creation of new data types. Here are four primitive data types supported by C++:

Data type	**Size**	**Range**
bool	1 byte	true or false
char	1 byte	-128 to 127
int	4 bytes	-2147483648 to 2147483647
double	8 bytes	$\pm 1.7 \times 10^{308}$

All data requires space in order to be stored. The fundamental unit of space for computers is known as the byte. A byte can represent 256 possible values. The more bytes you have, the more values you can represent. C++ uses 4 bytes for the "int" data type; the data type used for representing integers. Since 1 byte can represent 1 of 256 possible values, 4 bytes can represent 256^4 or approximately 4 billion possible values. Since the "int" data type should also store negative numbers, C++ uses approximately 2 billion of it's possible values for the negative integer values and approximately 2 billion of it's other possible values for the positive integer values.

The "char" data type represents characters which are symbols used primarily in human language. 1 byte seems good enough to store the alphabet of almost any one human language at a time, including any punctuation symbols and whitespace.

The "bool" data type represents only two values: true or false. A value of this data type is usually produced as the result of comparing two values. It is more useful than you think.

The "double" data type represents real numbers. Unlike integers, a double can represent fractional values and irrational numbers. If you are doing any scientific or financial calculations, you should use double as opposed to int.

Variables

Every variable has two things: a name and a value. To declare a variable in C++, you must specify the data type and the name of your new variable, you may optionally assign an initial value.

Listing 1: A series of declaration statements in C++

```
int x;
double amount;
bool cointoss;
int weight = 200;
double balance = 3070.12;
char c = 'A';
bool verified = true;
```

Variable names in C++ can only contain alphanumeric characters (excluding spaces) and cannot begin with a digit character. You may include underscores ('_') in your variable names. After declaring a variable, you may optionally assign a value to it with the '=' symbol (In C++, the '=' symbol is known as the assignment operator). If you do not assign a value, then the default value will be zero or false.

Expressions

An expression returns a value. Very often an expression may perform one or more of these actions: create a value, retrieve the value of a variable, assign a value to a variable, perform a calculation between one or two expressions.

There are many types of expressions. For now, the ones you should learn about are: literal values, assignment expressions, arithmetic expressions, and I/O expressions.

A literal value is a value that is not stored in a variable. Here are examples of literal values and their data types:

Data Type	Literal values
bool	true, false
char	'B', '5', '@', ' '
int	10, 7, 800, -23, 0, -46
double	459.54, 0.0, -1.34, 9.81e+2, -500.0e-6
const char*	"hello", "As above, so below"

The last data type literally reads "constant character pointer", it is the data type of a sequence of characters. You will learn more about them later.

An assignment expression assigns a value to a variable, the new value of the variable is the result of the expression as a whole. Here are a few assignment expressions:

Listing 2: Examples of assignment expressions

```
x = 4;
foo = bar;
a = b = c = d;
```

The first expression assigns a literal value 4 to the variable x. The second expression assigns the value inside variable bar to the variable foo. The third expression assigns the value d to variables a, b, and c.

The multiple assignments performed in the third assignment demonstrates how the result of one expression can be used inside another expression. The assignment `c = d` returns the new value of c, which was used as the value to be assigned to variable b, the new value of b was then used to assign to variable a. The variable d is itself an expression that returns the value stored in variable d. So what we have here are three sub-expressions inside one whole expression.

Arithmetic expressions create values by performing a calculation on one or more values (or sub-expressions). Arithmetic expressions make use of any of the arithmetic operators:

Operator Name	Symbol
Addition	`+`
Subtraction	`-`
Multiplication	`*`
Division	`/`
Modulo	`%`

Each of the above operators require two operands, one on the left-hand side of the operator and the other on the right-hand side of the operator. So the expression `a + b` uses the addition operator and the left-operand is the variable a and the right-operand is the variable b.

These operators should be self-explanatory, they can be used with the `int` or `double` data types. The modulo operator is similar to the division operator except for the fact that it returns the remainder instead of the quotient of the division.

An I/O expression is not a formally recognized expression type, but it can be suitable for your learning. An I/O expression uses one of these operators:

Operator Name	Symbol
Insertion	`<<`
Extraction	`>>`

I/O stands for Input/Output, it is related to data transfer on a stream. In C++, there are two data types related to streams.

Data Type	Description
istream	Input stream
ostream	Output stream

An ostream represents a destination to continually write data into. An ostream can be used to write data to a computer's display, or a file, or a network connection, just to name a few examples. All C++ programs can use a pre-declared variable known as `cout` in order to write to the program's standard output stream; for CLI programs, writing to `cout` usually results in data being displayed in the terminal window.

An istream represents a source to continually read data from until no more data is available. An istream can be used to read data from the computer's keyboard, or a file, or a network connection. The data read from an istream can be stored in variables. All C++ programs can use a pre-declared variable known as `cin` in order to read from the program's standard input stream; for CLI programs, reading from `cin` usually results in data being read from the computer's keyboard.

The result of an I/O expression is the istream or ostream variable that you are working with, therefore you may chain the I/O operators to read or write multiple values on the

stream.

The Hello World program

Now we that know all of that theory, let us take a look at a very simple program: the hello world program:

Listing 3: hello.cpp

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

int main()
{
    cout << "hello world" << endl;
    return 0;
}
```

This is what we call the source code of a computer program. Source code is a human-readable version of a computer program's algorithm. We must convert source code into a computer program before we can execute it (a process known as compiling), this is done by using a program known as a compiler.

Before we compile hello.cpp lets walk through what each line does.

Line 1 is not really a statement for our program, it is a statement for the preprocessor. The preprocessor is a special program that modifies our source code before it actually gets compiled. `#include <iostream>` means take all the code from the iostream file and paste it at the current line in our source code. iostream is a file that comes pre-packaged with any C++ distribution, it includes declarations of the `cin` and `cout` variables which we will use to perform standard I/O.

Line 2 is a special statement that's related to namespaces. A namespace is like a folder that you can declare all your variables inside of. All built-in C++ variables, including `cin` and `cout`, are declared inside the "std" namespace. To access a variable inside a namespace requires that you type the name of the namespace, followed by two colons, followed by the variable name. `using namespace std;` means to take all variables inside the std namespace and bring them out into the global, unnamed namespace, where they can be accessed without referring to std. Without this statement you would have to type `std::cout` everytime you needed to use `cout`. For this course, you should always include this statement in your program.

`int main()` is the declaration of a function. A function, which you will learn more about later, is a sequence of statements which the program can execute at any time. In C++, the function labeled "main" is the entry point of every program, it is the first function that gets executed.

The statements that go inside the main function are enclosed in { and }. Statements enclosed inside matching curly braces are known as compound statements or code blocks. `cout << "hello world" << endl;` makes use of the insertion operator in order to write two values into the program's standard output. The values written include the literal value "hello world", whose data type is `const char*`, and the variable `endl` which writes a new-line character into any ostream it is written to.

`return 0;` exits the main function with the value 0. This value is the exit code for our program. 0 means success, a non-zero value means the program failed.

Compiling hello.cpp

Compilers as well as other programmer tools can be executed through the operating system's command-line interface.

Assuming that you are using Linux, if you saved hello.cpp in your Documents folder, then you can compile it by typing these commands into your terminal emulator:

```
cd ~/Documents
c++ hello.cpp
```

A program named "a.out" will be produced. You may execute the a.out program with the following command:

```
./a.out
```

Take note that the compiler and the hello world program both execute within an environment known as the command-line interface. This environment has text-only graphics; devoid of any buttons, scrollbars, or images. This is a programmer-friendly environment that is not only easier to create programs in, but also extends more power to the user.