C++ Lecture 1
Intro to C++

CSci 588: Data Structures, Algorithms and Software Design

http://www.cplusplus.com/

http://en.cppreference.com/w/

Overview

- Program Structure
- Variables
- Constants
- Operators
- Basic Input/Output

Program Structure

Program Structure

```
// my first program in C++

#include <iostream>
using namespace std;

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

```
Hello World!
```

On the left is a simple Hello World! program, and on the right is what it will output when compiled and executed.

Program Structure

```
// my first program in C++

#include <iostream>
using namespace std;

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

This is a comment line. Anything on a line after a // is ignored by the compiler and has no effect on the program.

Program Structure

```
// my first program in C++

#include <iostream>
using namespace std;

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

Any line beginning with a # is a *preprocessor directive*. These are special commands for the compiler and are not regular C or C++ lines of code.

This directive tells the compiler to include the iostream standard file, which has all the declarations for the basic standard input-output library of C++.

#include preprocessor directives have to be at the top of the files they're used in.

Program Structure

```
// my first program in C++

#include <iostream>
using namespace std;

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

Everything in the standard C++ library is declared in a namespace called 'std'. This is a very common statement in C++ programs, as many use the standard library (std) and use it to have access to the functionality in it.

Program Structure

```
// my first program in C++

#include <iostream>

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


This is the same as the previous code, except we're explicitly specifying that cout and endl are from the namespace std.

Program Structure

```
// my first program in C++

#include <iostream>
using namespace std;

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

The main function is the first function that gets called when your C++ program is run. The main function can be anywhere within your code. Every C++ program requires a main function and you can't have more than one.

The *int* before 'main' is the return type of the function, and is short for integer. This means the main function must return an integer somewhere.

The parenthesis after 'main' are required for every function. Within the parenthesis are the parameters to the function. In this case, there are none.

Following the parameters is a { which signifies the beginning of the function. The } signifies the end of the function. {}s and the code between them is typically called a *block* of code.

Program Structure

```
// my first program in C++

#include <iostream>
using namespace std;

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

This line is a C++ statement (statements typically end in a ';'). Statements contain an expression or compound expression and actually produce some effect.

'cout' is the name of the standard output stream in C++ (what will write to your terminal screen or command prompt).

'<<' is an operator which inserts sequences of characters (in this case "Hello World!") into a stream.

'cout' is declared in the iostream standard file within the std namespace, which is why we needed those lines at the top of the program.

Program Structure

```
// my first program in C++

#include <iostream>
using namespace std;

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

A return statement causes a function to finish. In this case, because the main function needs to return an integer, we return 0, which for the main function typically means the program worked as expected without any errors.

Most C++ console programs end this way.

Program Structure

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

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

These two blocks of code are identical. Unlike some other programming languages (like python) where the tabs and whitespace have semantic meaning, in C++ the whitespace between statements is just ignored and has no meaning.

Program Structure

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

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

This doesn't mean write code like on the bottom, because it is much more difficult to understand and debug.

Program Structure

```
// my second program in C++

#include <iostream>

using namespace std;

int main ()
{
  cout << "Hello World! ";
  cout << "I'm a C++ program";
  return 0;
}
```

```
Hello World! I'm a C++ program
```

The above modified program prints out the text on the left (note that they are on the same line).

Program Structure

```
int main ()
{
  cout << "Hello World! ";
  cout << "I'm a C++ program";
  return 0;
}
```

```
int main ()
{
  cout <<
    "Hello World! ";
  cout
    << "I'm a C++ program";
  return 0;
}
```

```
int main () { cout << " Hello World! "; cout << " I'm a C++ program "; return 0; }
```

As before, all three of these main methods work exactly the same way. Differences in whitespace are ignored.

Program Structure

```
int main ()
{
  cout << "Hello World! ";
  cout << "I'm a C++ program";
  return 0;
}
```

```
int main ()
{
  cout <<
    "Hello World! ";
  cout
    << "I'm a C++ program";
  return 0;
}
```

```
int main () { cout << " Hello World! "; cout << " I'm a C++ program "; return 0; }
```

Again, don't do this (you'd be surprised what some programmers do).

Comments

There are two ways to have comments in C++ code:

```
// line comment
/* block comment */
```

Comments

```
/* my second program in C++
   with more comments. block
   comments can span multiple
   lines! */

#include <iostream>
using namespace std;

int main ()
{
  cout << "Hello World! ";     // prints Hello World!
  cout << "I'm a C++ program"; // prints I'm a C++ program
  return 0;
}
```

This program prints out the same output as the previous ones.

The text in green are all comments. Block comments can span multiple lines, while line comments are only from the // to the end of the line.

Variables

Variables

```
int a = 5;
int b = 2;
a = a + 1;
int result = a - b;
```

This code defines three variables: *a*, *b* and *result*.

Variables are used to store information for later use.

Each variable has not only a name, or *identifier*, (in this case *a*, *b*, or *result*) but also a type. In this case they are all integers.

Identifiers

An identifier consists of only characters, numbers and underscores ('_'). An identifier must also begin with either an underscore or character.

Identifiers also cannot be *reserved keywords* (terms in the C++ programming language). The standard reserved keywords are:

```
asm, auto, bool, break, case, catch, char, class, const, const_cast,
continue, default, delete, do, double, dynamic_cast, else, enum, explicit,
export, extern, false, float, for, friend, goto, if, inline, int, long,
mutable, namespace, new, operator, private, protected, public, register,
reinterpret_cast, return, short, signed, sizeof, static, static_cast,
struct, switch, template, this, throw, true, try, typedef, typeid, typename,
union, unsigned, using, virtual, void, volatile, wchar_t, while
```

Also, some alternative representations for operators cannot be used:

```
and, and_eq, bitand, bitor, compl, not, not_eq, or, or_eq, xor, xor_eq
```

There may also be additional reserved keywords that are specific to your compiler.

Identifiers

C++ is also **case sensitive** -- that means these three variables:

```
int myVariable;
int myvariable;
int MYVARIABLE;
```

Are all different variables.

Fundamental Data Types

Name	Description	Size*	Range*
`char`	Character or small integer.	1byte	signed: -128 to 127 unsigned: 0 to 255
`short int` (`short`)	Short Integer.	2bytes	signed: -32768 to 32767 unsigned: 0 to 65535
`int`	Integer.	4bytes	signed: -2147483648 to 2147483647 unsigned: 0 to 4294967295
`long int` (`long`)	Long integer.	4bytes	signed: -2147483648 to 2147483647 unsigned: 0 to 4294967295
`bool`	Boolean value. It can take one of two values: true or false.	1byte	`true` or `false`
`float`	Floating point number.	4bytes	+/- 3.4e +/- 38 (~7 digits)
`double`	Double precision floating point number.	8bytes	+/- 1.7e +/- 308 (~15 digits)
`long double`	Long double precision floating point number.	8bytes	+/- 1.7e +/- 308 (~15 digits)
`wchar_t`	Wide character.	2 *or* 4 bytes	1 wide character

*Size and Range are architecture dependent. The values here are for 32 bit systems. For other systems, the size of an int is one word (usually 32 or 64 bits). `char`, `short`, `int` and `long` each must be at least larger than the previous, and the same for `float`, `double` and `long double`.

Declaring Variables

```
int a;
float mynumber;


int a, b, c;

int a;
int b;
int c;
```

```
unsigned short int NumberOfSisters;
signed int MyAccountBalance;


int MyAccountBalance;


short Year;
short int Year;


unsigned NextYear;
unsigned int NextYear;
```

Declaring Variables

```
int a;
float mynumber;
```

To declare a variable, you need to specify at least a name and a type. Each of these is a statement (they end with a semicolon ';').

```
int a, b, c;
```

```
int a;
int b;
int c;
```

```
unsigned short int NumberOfSisters;
signed int MyAccountBalance;
```

```
int MyAccountBalance;
```

```
short Year;
short int Year;
```

```
unsigned NextYear;
unsigned int NextYear;
```

Declaring Variables

```
int a;
float mynumber;
```

```
int a, b, c;
```

You can declare multiple variables of the same type using commas (‘,’) in between each name/identifier.

```
int a;
int b;
int c;
```

```
unsigned short int NumberOfSisters;
signed int MyAccountBalance;
```

```
int MyAccountBalance;
```

```
short Year;
short int Year;
```

```
unsigned NextYear;
unsigned int NextYear;
```

Declaring Variables

```
int a;
float mynumber;
```

```
int a, b, c;
```

```
int a;
int b;
int c;
```

These three statements have the same meaning as the previous statement.

```
unsigned short int NumberOfSisters;
signed int MyAccountBalance;
```

```
int MyAccountBalance;
```

```
short Year;
short int Year;
```

```
unsigned NextYear;
unsigned int NextYear;
```

Declaring Variables

```
int a;
float mynumber;
```

```
int a, b, c;
```

```
int a;
int b;
int c;
```

```
unsigned short int NumberOfSisters;
signed int MyAccountBalance;
```

```
int MyAccountBalance;
```

```
short Year;
short int Year;
```

```
unsigned NextYear;
unsigned int NextYear;
```

`int`, `short`, and `long` variables are `signed` by default (meaning they can have both positive and negative values). You can also specify it explicitly.

It is also possible to have `unsigned int`, `short` and `long` variables, which cannot be negative (but have a larger maximum range).

Declaring Variables

```
int a;
float mynumber;
```

```
int a, b, c;
```

```
int a;
int b;
int c;
```

```
unsigned short int NumberOfSisters;
signed int MyAccountBalance;
```

```
int MyAccountBalance;
```

Again, `short`, `int` and `long` are signed by default. So this statement is equivalent to the previous one.

```
short Year;
short int Year;
```

```
unsigned NextYear;
unsigned int NextYear;
```

Declaring Variables

```
int a;
float mynumber;
```

```
int a, b, c;
```

```
int a;
int b;
int c;
```

```
unsigned short int NumberOfSisters;
signed int MyAccountBalance;
```

```
int MyAccountBalance;
```

```
short Year;
short int Year;
```

`short` is equivalent to `short int`, and `long` is equivalent to `long int`. These two statements are the same.

```
unsigned NextYear;
unsigned int NextYear;
```

Declaring Variables

```
int a;
float mynumber;
```

```
int a, b, c;
```

```
int a;
int b;
int c;
```

```
unsigned short int NumberOfSisters;
signed int MyAccountBalance;
```

```
int MyAccountBalance;
```

```
short Year;
short int Year;
```

```
unsigned NextYear;
unsigned int NextYear;
```

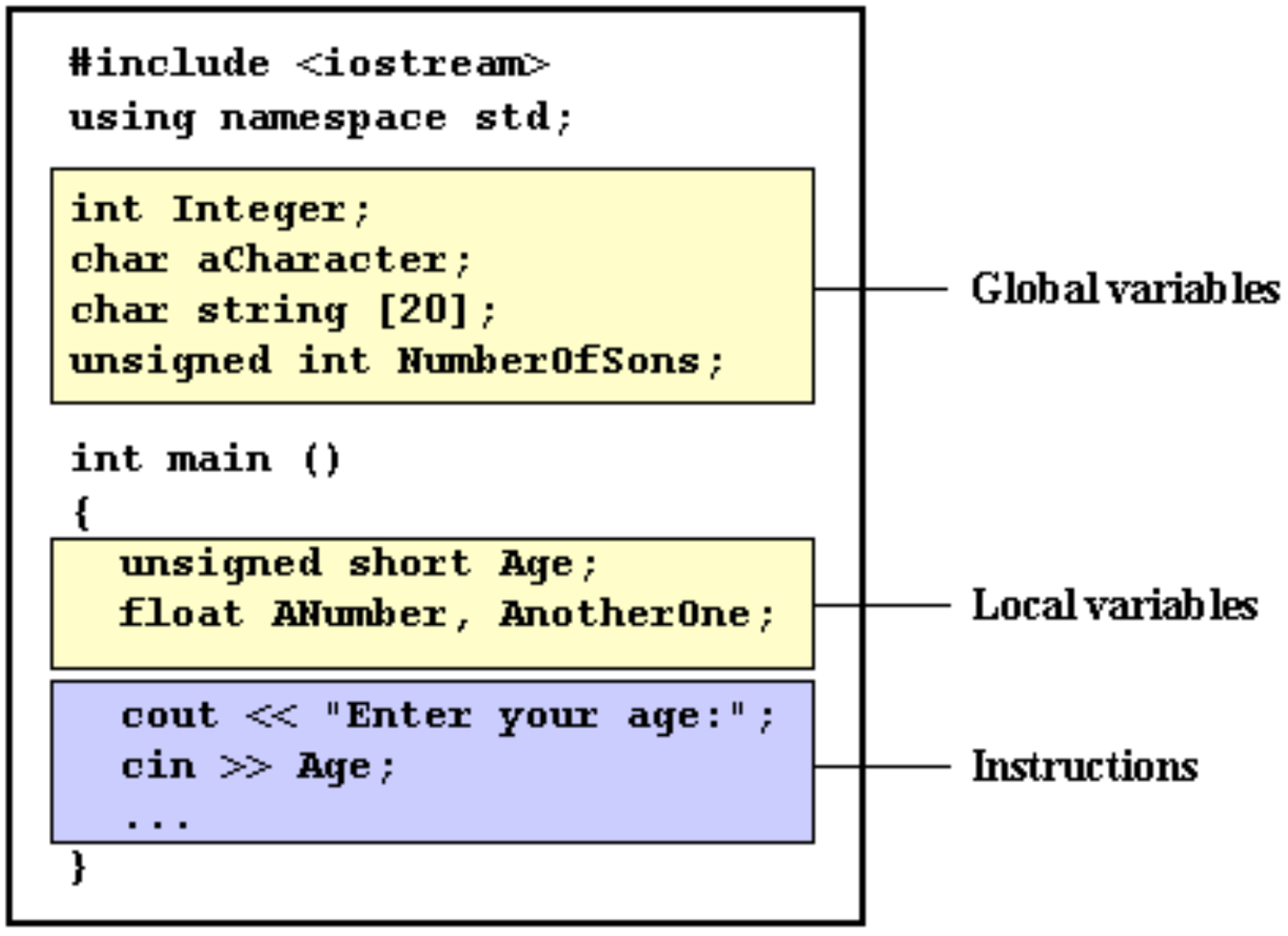
`signed` is equivalent to `signed int`, and `unsigned` is equivalent to `unsigned int`. These two statements are the same.

Declaring Variables

```
// operating with variables

#include <iostream>
using namespace std;

int main ()
{
  // declaring variables:
  int a, b;
  int result;

  // process:
  a = 5;
  b = 2;
  a = a + 1;
  result = a - b;

  // print out the result:
  cout << result;

  // terminate the program:
  return 0;
}
```

The output of the above program is 4.

Scope of Variables



Variables have *scope*. *Global variables* can be used throughout the entire file. *Local variables* can be used only on lines in the block they have been declared, and only on lines after they have been declared.

Scope of Variables

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

int a = 15;

int main ()
{
  // declaring variables:
  int a;

  a = 5;
  {
    int a;
    a = 10;
    cout << a << “\n”;
  }
  cout << a << “\n”;

  // terminate the program:
  return 0;
}
```

Variables can be redeclared. What does this program output?

Scope of Variables

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

int a = 15;

int main ()
{
  // declaring variables:
  int a;

  a = 5;
  {
    int a;
    a = 10;
    cout << a << “\n”;
  }
  cout << a << “\n”;

  // terminate the program:
  return 0;
}
```

This outputs:

10

5

Scope of Variables

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

int a = 15;

int main ()
{
  // declaring variables:
  int a;

  a = 5;
  {
    int a;
    a = 10;
    cout << a << “\n”;
  }
  cout << a << “\n”;

  // terminate the program:
  return 0;
}
```

The variable a was redeclared within the main function, and again in the inner block, and its scope is only that inner block. Outside the inner block the identifier a refers to the first declaration in main (which overrides the global declaration).

Initializing Variables

```
int a = 0; // c-like initialization

int a (0); // constructor initialization
```

There are two ways to initialize variables. *c-like initialization* (which is what the c programming language uses) and *constructor initialization*. Both of these are equivalent.

Initializing Variables

```
// initialization of variables

#include <iostream>
using namespace std;

int main ()
{
  int a=5;               // initial value = 5
  int b(2);              // initial value = 2
  int result;            // initial value undetermined

  a = a + 3;
  result = a - b;
  cout << result;

  return 0;
}
```

This program outputs:
6

Terminology

Blocks

Statement

Expression

Variable

Identifier

Scope

Blocks

A block of code is everything between a pair of *matching* { }*s:*

```
int main ()
{
  // declaring variables:
  int a;

  a = 5;
  {
    int a;
    a = 10;
    cout << a << “\n”;
  }
  cout << a << “\n”;

  // terminate the program:
  return 0;
}
```

Blocks

A block of code is everything between a pair of *matching* { }*s:*

This is a block

```
int main ()
{
  // declaring variables:
  int a;

  a = 5;
  {
    int a;
    a = 10;
    cout << a << “\n”;
  }
  cout << a << “\n”;

  // terminate the program:
  return 0;
}
```

Blocks

A block of code is everything between a pair of *matching* { }*s:*

This is also a block

```
int main ()
{
  // declaring variables:
  int a;

  a = 5;
  {
    int a;
    a = 10;
    cout << a << “\n”;
  }
  cout << a << “\n”;

  // terminate the program:
  return 0;
}
```

Matching {}s and ()s

Note that some of the most common problems for beginning programmers are not matching {}s and ()s appropriately.

Properly formatting your code and entering the closing } or) when you type the opening { or) is the most effective way to prevent this from happening.

Statements

A statement is a line of code (anything between ;s), a loop statement, or a conditional statement.

Identifiers

Identifiers are names you choose for variables and functions:

```
int square_input (int input) {
  return input * input;
}

int main () {
  // declaring variables:
  int a = 5;
  int b = square_input(a);

  cout << a << “\n”;

  // terminate the program:
  return 0;
}
```

Everything in bold is an identifier (main, a, b, input).

Variable

Variables are identifiers which represent some value (and its declared *type*) which changes over time (hence the name variable).

```
int double_input (int input) {
  return input * input;
}

int main () {
  // declaring variables:
  int a = 5;
  int b = double_input(a);
  a = a * b;

  cout << a << “, “ << b << “\n”;

  // terminate the program:
  return 0;
}
```

Scope

The scope of a variable is where in the program you can use that variable.

```
int a = 15;
int c = 25;

int increment(int input) {
  return input + 1;
}

int main () {
  // declaring variables:
  int b = 5;
  if (b < 4) {
    int c = increment(b);
    a = 10;
    cout << a << “\n”;
  }
  cout << a << “\n”;

  // terminate the program:
  return 0;
}
```

Scope

The scope of a variable is where in the program you can use that variable.

‘a’ is a global variable and it’s scope is the entire program (everywhere after it is declared).

```
int a = 15;
int c = 25;

int increment(int input) {
  return input + 1;
}

int main () {
  // declaring variables:
  int b = 5;
  if (b < 4) {
    int c = increment(b);
    a = 10;
    cout << a << “\n”;
  }
  cout << a << “\n”;

  // terminate the program:
  return 0;
}
```

Scope

The scope of a variable is where in the program you can use that variable.

‘c’ is also a global variable and it’s scope is the entire program (everywhere after it is declared).

```
int a = 15;
int c = 25;

int increment(int input) {
  return input + 1;
}

int main () {
  // declaring variables:
  int b = 5;
  if (b < 4) {
   int c = increment(b);
    a = 10;
    cout << a << “\n”;
  }
  cout << a << “\n”;

  // terminate the program:
  return 0;
}
```

Scope

The scope of a variable is where in the program you can use that variable.

However you can't use 'c' before it is declared.

```
int a = 15 * c;
int c = 25;

int increment(int input) {
  return input + 1;
}

int main () {
  // declaring variables:
  int b = 5;
  if (b < 4) {
    int c = increment(b);
    a = 10;
    cout << a << "\n";
  }
  cout << a << "\n";

  // terminate the program:
  return 0;
}
```

Scope

The scope of a variable is where in the program you can use that variable.

The scope of the variable ‘input’ (and any other variables in a function declaration) are only within that function.

```
int a = 15 * c;
int c = 25;

int increment(int input) {
  return input + 1;
}

int main () {
  // declaring variables:
  int b = 5;
  if (b < 4) {
    int c = increment(b);
    a = 10;
    cout << a << “\n”;
  }
  cout << a << “\n”;

  // terminate the program:
  return 0;
}
```

Scope

The scope of a variable is where in the program you can use that variable.

This means you cannot use that ‘input’ variable outside of the function.

```
int a = 15 * c;
int c = 25;

int increment(int input) {
  return input + 1;
}

int main () {
  // declaring variables:
  int b = 5;
  if (b < input) {
    int c = increment(b);
    a = 10;
    cout << a << “\n”;
  }
  cout << a << “\n”;

  // terminate the program:
  return 0;
}
```

Scope

The scope of a variable is where in the program you can use that variable.

The scope of the variable ‘b’ (and any other variables in a function declaration) are only within that function, after the variable has been declared. main is also function.

```
int a = 15 * c;
int c = 25;

int increment(int input) {
  return input + 1;
}

int main () {
  // declaring variables:
  int b = 5;
  if (b < 4) {
    int c;
    a = 10;
    cout << a << “\n”;
  }
  cout << a << “\n”;

  // terminate the program:
  return 0;
}
```

Scope

The scope of a variable is where in the program you can use that variable.

The *actual* definition is that the scope of a variable is anywhere after its declaration within the *block* it was declared in.

```
int a = 15 * c;
int c = 25;

int increment(int input) {
  return input + 1;
}

int main () {
  // declaring variables:
  int b = 5;
  if (b < 4) {
    int c;
    a = 10;
    cout << a << “\n”;
  }
  cout << a << “\n”;

  // terminate the program:
  return 0;
}
```

Scope

The scope of a variable is where in the program you can use that variable.

So that means the scope of c is within the block of the if statement only.

```
int a = 15 * c;
int c = 25;

int increment(int input) {
  return input + 1;
}

int main () {
  // declaring variables:
  int b = 5;
  if (b < 4) {
    int c;
    a = 10;
    cout << a << “\n”;
  }
  cout << a << “\n”;

  // terminate the program:
  return 0;
}
```

Scope

The scope of a variable is where in the program you can use that variable.

This means you can't use 'c' outside of that block, even after it has been declared.

```
int a = 15 * c;
int c = 25;

int increment(int input) {
  return input + 1;
}

int main () {
  // declaring variables:
  int b = 5;
  if (b < 4) {
    int c;
    a = 10;
    cout << a << "\n";
  }
  cout << c << "\n";

  // terminate the program:
  return 0;
}
```