School of Computing
FACULTY OF ENGINEERING

Introduction to C++ Programming

Day 1: Fundamentals of C++

Nick Efford

Email: N.D.Efford@leeds.ac.uk
Twitter: @python33r
Google+: http://gplus.to/pythoneer

Welcome!

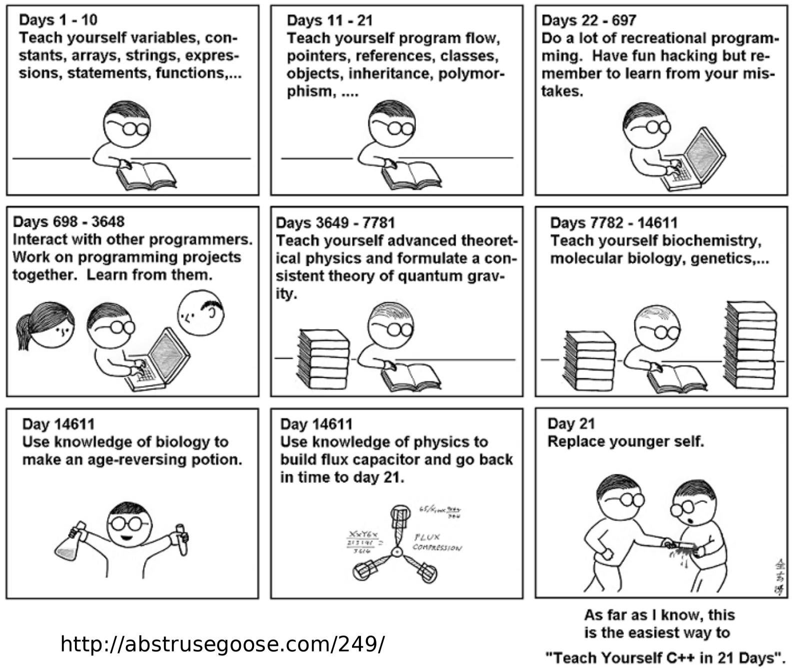
On this course, you will

- Be introduced to the most important bits of C++
- Have the opportunity to practice using some of these features in C++ programs

You will <u>not</u>

- Become expert C++ programmers
- Learn about platform-specific development tools

Note that we assume prior experience of programming in another language...


Days 1 - 10
Teach yourself variables, constants, arrays, strings, expressions, statements, functions,...
Days 11 - 21
Teach yourself program flow, pointers, references, classes, objects, inheritance, polymorphism, ...
Days 22 - 697
Do a lot of recreational programming. Have fun hacking but remember to learn from your mistakes.
Days 698 - 3648
Interact with other programmers. Work on programming projects together. Learn from them.
Days 3649 - 7781
Teach yourself advanced theoretical physics and formulate a consistent theory of quantum gravity.
Days 7782 - 14611
Teach yourself biochemistry, molecular biology, genetics,...
Day 14611
Use knowledge of biology to make an age-reversing potion.
Day 14611
Use knowledge of physics to build flux capacitor and go back in time to day 21.
FLUX COMPRESSION
Day 21
Replace younger self.
As far as I know, this is the easiest way to "Teach Yourself C++ in 21 Days".

http://abstrusegoose.com/249/

Course Structure

Mornings:

- Mainly me showing you things...
- ...and you asking questions (hopefully)
- Small programming tasks or paper exercises
- 15-minute break for coffee, etc

Afternoons:

- Mainly you, doing more extended exercises
- Some presentation from me if needed

Materials are in the VLE: http://vlebb.leeds.ac.uk/

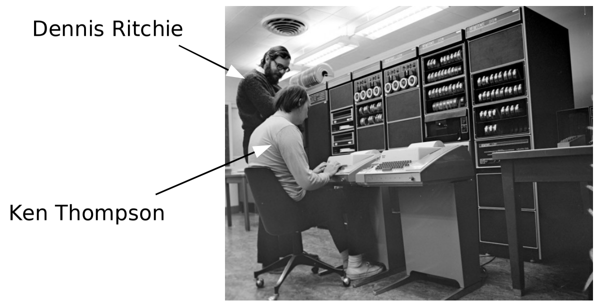
Today’s Objectives

- For you to acquire an understanding of the basic syntax and features of the C++ programming language
- For you to gain some experience of writing, compiling, running and debugging small C++ programs

Today’s Topics

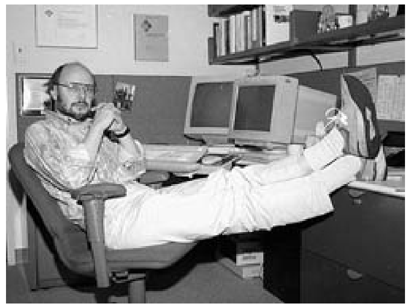
- Origins of C & C++
- Basic structure of a C++ program
- Compilation
- Primitive data types
- Defining variables and constants
- Operators and expressions
- Basic console I/O
- Selection: `if` statements
- Repetition: `while` & `for`
- Storage of multiple values in arrays

Origins of C



1972	Created by Dennis Ritchie, for UNIX development
1978	Publication of ‘K&R’ book (→ ‘K&R C’)
1989	ANSI (later ISO) standardisation (→ ‘C89’)
1999	Revision of ISO standard (→ ‘C99’)

Origins of C++



1979	Bjarne Stroustrup at Bell Labs, ‘C with Classes’
1983	First version of C++ used internally by AT&T
1985	First commercial C++ development tools
1998	ISO standardisation

C++ vs. C

- Better type checking than C
- Supports a wider range of programming styles
  - Object-oriented programming
  - Generic programming
- Retains almost all of C as a subset
- Bigger and <u>much</u> more complex than C!
- Both are widely used in industry

"Hello, World!" in C++

Header, included whenever we need to do input/output

```
#include <iostream>

int main()
{
  std::cout << "Hello, World!" << std::endl;

  return 0;
}
```

Execution always starts in a **function** called `main`

value returned to caller – in this case, the environment running the program

"Hello, World!" in C++

object representing <u>c</u>onsole <u>out</u>put stream

```
std::cout << "Hello, World!" << std::endl;
```

stream insertion operator

stream manipulator that ends the line, then flushes output buffer

Slightly Simplified Version

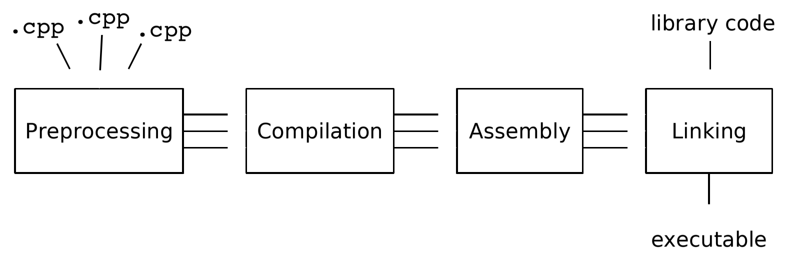
```
#include <iostream>

using namespace std;

int main()
{
  cout << "Hello, World!" << endl;

  return 0;
}
```

allows us to omit `std::` prefix from `cout` & `endl` (more on this later...)

Exercise 1: create this file in a text editor...

‘Compiling’ C++ Programs

- Multi-stage process
- One or more C++ **source files** as input
- Single output file containing native **machine code**, executable directly on computer’s CPU



‘Compiling’ C++ Programs

1. Preprocessing
   - Inclusion of **header files**
   - Definition of **macros**
   - Conditional compilation
2. Compilation
   - Source code translated to **assembly language**
3. Assembly
   - Assembly language ‘assembled’ into **object code**
4. Linking
   - Files of object code combined with library code to create an **executable**

The GNU Compiler Collection

- Free, from http://gcc.gnu.org/
- Standard on Linux
- Available for Mac OS X in Xcode toolset
- Available for Windows
  - MinGW, www.mingw.org
  - As part of **Cygwin**, www.cygwin.com

The GNU C++ Compiler

Command syntax

```
g++ [options] source-file [source-file...]
```

```
-o      Specifies output filename
-Wall   Turn on (nearly) all compiler warnings
-g      Generate debugging information
-c      Generate object code but don’t link
```

Example

```
g++ -Wall -g -o hello hello.cpp
```

C++’s Primitive Data Types

Numeric

- Integer
- Floating-point

Non-numeric

- Character
- Boolean

Primitive Numeric Types: Full Names

```
signed char
signed short int
signed int
signed long int

unsigned char
unsigned short int
unsigned int
unsigned long int

float
double
long double
```

Characters can be treated as integers!

Primitive Numeric Types: Usual Names

```
char
short
int
long

unsigned char
unsigned short
unsigned int
unsigned long

float
double
long double
```

What range of values can these types represent?

How much storage do they require?

Data Representation: Ranges

Intel® Core^{TM}2 Quad Q9400 (64-bit)

```
$ ./ranges
signed char    : -128 to 127
signed short   : -32768 to 32767
signed int     : -2147483648 to 2147483647
signed long    : -9223372036854775808 to 9223372036854775807
unsigned char  : 0 to 255
unsigned short : 0 to 65535
unsigned int   : 0 to 4294967295
unsigned long  : 0 to 18446744073709551615
```

Intel® Atom N270 (32-bit)

```
$ ./ranges
signed char    : -128 to 127
signed short   : -32768 to 32767
signed int     : -2147483648 to 2147483647
signed long    : -2147483648 to 2147483647
unsigned char  : 0 to 255
unsigned short : 0 to 65535
unsigned int   : 0 to 4294967295
unsigned long  : 0 to 4294967295
```

Data Representation: Sizes

Intel® Core^{TM}2 Quad Q9400 (64-bit)

```
$ ./sizes
sizeof(char)           = 1
sizeof(short)          = 2
sizeof(int)            = 4
sizeof(long)           = 8
sizeof(unsigned char)  = 1
sizeof(unsigned short) = 2
sizeof(unsigned int)   = 4
sizeof(unsigned long)  = 8

sizeof(float)          = 4
sizeof(double)         = 8
sizeof(long double)    = 16
```

Intel® Atom N270 (32-bit)

```
$ ./sizes
sizeof(char)           = 1
sizeof(short)          = 2
sizeof(int)            = 4
sizeof(long)           = 4
sizeof(unsigned char)  = 1
sizeof(unsigned short) = 2
sizeof(unsigned int)   = 4
sizeof(unsigned long)  = 4

sizeof(float)          = 4
sizeof(double)         = 8
sizeof(long double)    = 12
```

Exercise 3: compile and run `ranges` and `sizes` on your PC...

Primitive Non-Numeric Types

Characters

- `char` (8-bit)
- `wchar_t` (16-bit)

Boolean values

- `bool` (values `true` & `false`)

Literal Values

Decimal `int`	`42`
Octal (base 8) `int`	`052`
Hexadecimal (base 16) `int`	`0x2a`
`long`	`42L`
`unsigned int`	`42U`
`unsigned long`	`42UL`
`float`	`41.99F`
`double`	`41.99, 5.67e-3`
`long double`	`41.99L`
`char`	`'x', '\167', '\n'`
`wchar_t`	`L'x'`

Variables & Assignment

Syntax

type variable-name ;
type variable-name = *initial-value* ;

Examples

```
int x;
int y = 42;
float z = 1.073f;
```

- You are not required to initialise variables before use
- Value of an uninitialised variable is <u>undefined</u>
- Variables can be defined anywhere within a code block

Constants

Use the `const` keyword:

```
const int maxSize = 100;

const double lightSpeed = 2.99792e+8;
```

value <u>must</u> be supplied at definition time

Attempting to assign a new value to a constant triggers a compiler error...

Quiz

1. How would you define mathematical constant π ?
2. Write down a definition for a long integer variable named `counter` with an initial value of zero
3. What is the result of the following code?

```
short size = 100000;
```

A. Assignment of 100,000 to variable `size`
B. Compiler error
C. Compiler warning and some other value for `size`
D. Run-time error when the statement executes

4. Does the statement `int x = 2.5;` compile?

Answers

Strings in C++

- Made available via `#include <string>`
- Literal values delimited by "
- Characters accessed using `[]` and <u>zero-based</u> index
- Can be compared using ==, <, etc
- Can be concatenated using +, +=, etc

```
string message = "Hello, ";
message += "World!";
cout << message << endl;
cout << message[4] << endl;
```

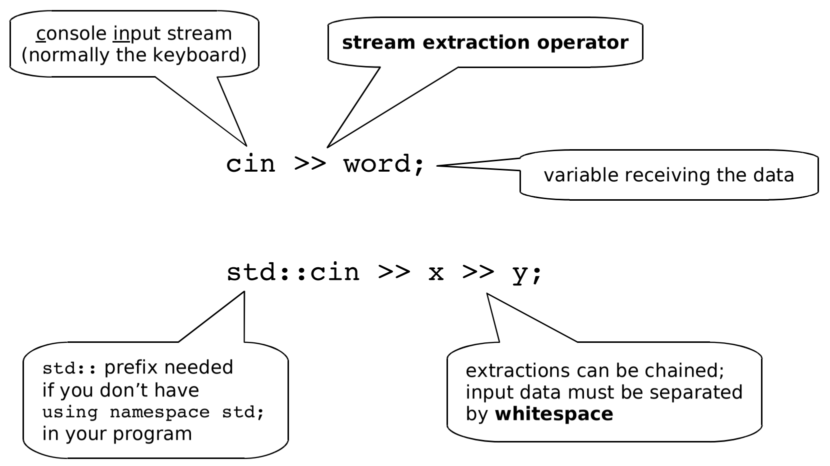
Useful String Operations

`length`	Returns length of the string
`append`	Appends a string or chars to the string
`insert`	Inserts a string or chars into the string
`find`	Searches for a sequence of chars
`substr`	Returns part of this string as a new string
`replace`	Replaces a sequence of chars
`c_str`	Returns this string's characters as a C string

Example

```
#include <string>
#include <iostream>

using namespace std;

int main()
{
  string word;
  cout << "Enter a word: ";
  cin >> word;

  cout << "Word length = " << word.length() << endl;
  if (word.find("x") != string::npos)
    cout << "Your word contains an 'x'" << endl;

  return 0;
}
```

Reading Stuff:
Stream Extraction



```
cin >> word;

std::cin >> x >> y;
```

console input stream
(normally the keyboard)
stream extraction operator
variable receiving the data
std:: prefix needed if you don't have using namespace std; in your program
extractions can be chained; input data must be separated by **whitespace**

Operators & Expressions

Arithmetic:	`+ - * / %`
Relational:	`< <= > >= == !=`
Logical:	`&& || !`
Bitwise:	`& | ^`

Examples

```
x = y*(z - 1)/2;

n = (n + 1) % maximum;

bool a_smaller = a < b;
```

Operators & Expressions

Increment by 1:	++
Decrement by 1:	--
Add and assign:	+=
Subtract and assign:	-=

Examples

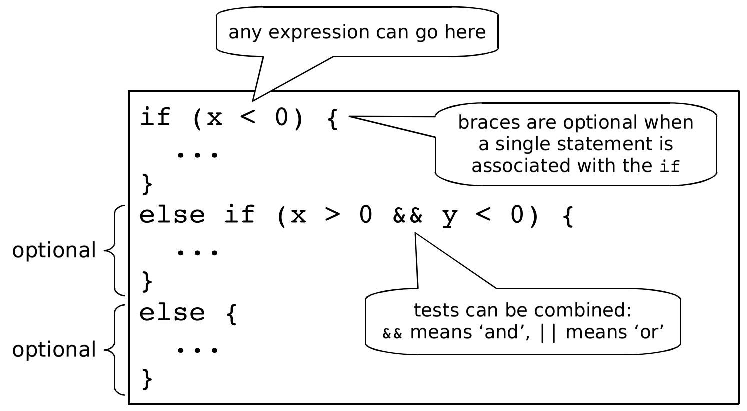
```
n = n + 1;    // increment n by 1
n += 1;       // shorter version
++n;          // shortest version
n++;
```

Quiz

1. Write statements defining `int` variables `x` and `y`, with initial values of 2 and 7, respectively
2. Write a statement that defines a float variable `average` and initialises it to the average of `x` and `y`

Answers

Making Decisions: `if` statements



```
if (x < 0) {
   ...
}
else if (x > 0 && y < 0) {
   ...
}
else {
   ...
}
```

any expression can go here
braces are optional when a single statement is associated with the if
tests can be combined: && means 'and', || means 'or'
optional
optional

Puzzle

```
if (x = 0)
  cout << "x is zero!" << endl;
else
  cout << "x is non-zero!" << endl;
```

What is printed when `x` has the value 1?

What is printed when `x` has the value 0?

Repetition with `while`

How do we compute the sum of the integers from 1 to 100?

```
int sum = 0;

int n = 1;
while (n <= 100) {
  sum += n;
  ++n;
}

cout << "Sum is " << sum << endl;
```

loop control variable

test whether loop should continue executing

`do...while`

```
int value;

do {
  cout << "Enter a positive integer: ";
  cin >> value;
}
while (value <= 0);
```

Loop is guaranteed to run at least once

`for` Loops

Syntax

```
for (set up control variable;
      condition for loop to continue;
      alter control variable) {
   ...
}
```

Example

```
for (int i = 10; i > 0; --i) {
  cout << i << endl;
}
cout << "Lift Off!" << endl;
```

Test Yourself

Rewrite the code on Slide 38 using a `for` loop:

Storing Collections of Values

- Datasets can consist of many values
- Representing each value with a variable is impractical and inflexible (not to mention tedious)
- We need a data structure that can hold multiple values
- We need easy access to values and we need it to be flexible – growing or shrinking in size on demand
- One solution is the **vector**, which becomes available to us if we include the vector header

```
#include <vector>
```

Creating Vectors

```
vector<int> a;               // empty vector of integers
vector<int> b(10);           // 10 integers, all 0
vector<int> c(5, -1);        // 5 integers, all -1
vector<int> d(c);            // copy of c

vector<string> p;            // empty vector of strings
vector<string> q(5, "xyz");  // 5 strings, all "xyz"
vector<string> r(q);         // copy of q
```

any type can go inside <>
– even another vector!

Adding & Removing Values

`push_back`	Adds item to end of vector
`pop_back`	Removes value from end of vector
`insert`	Inserts a value at a specified position
`erase`	Remove value(s) at specified position(s)
`clear`	Empties vector of all stored values

```
vector<string> words;
words.push_back("Hello");
words.push_back("World!");
cout << words.size() << endl;   // prints current size
words.pop_back();
cout << words.size() << endl;   // what will this print?
```

Accessing Vector Elements

```
vector<int> v;
for (int n = 1; n <= 5; ++n) {
  v.push_back(n*n);
}

cout << v.size() << endl;   // prints size of v
cout << v[0] << endl;       // prints first element of v
cout << v[4] << endl;       // prints last element of v
cout << v[5] << endl;       // what happens here?
cout << v.at(0) << endl;    // same as v[0]
cout << v.at(4) << endl;    // same as v[4]
cout << v.at(5) << endl;    // what happens here?
```

Summary

We have

- Reviewed the history and relevance of C++
- Dissected the process of compilation
- Examined the primitive data types available in C++
- Investigated the basic syntax of C++ programs
- Looked how console input/output is done
- Introduced the idea of **vectors** as a means of associating multiple values with a single variable

Follow-up Work

- Do this afternoon’s exercises, completing in your own time if necessary
- Read up on today’s topics in a C++ book
- Get GCC installed on your own PC