Crash Course in C++

R F L Evans

www-users.york.ac.uk/~rfle500/

Course overview

- Lecture 1 - Basic Programming
- hello world
- Variables and scope - int, double, float, bool
- Standard library
- string
- constructs - for, while, if

Additional resources

- www.cplusplus.com/doc/tutorial
- http://www.parashift.com/c++-faq/index.html
- http://www.agner.org

Lecture 1
Basic programming

- Introduction
- Hello world
- Variables and Operators
- The C++ standard library
- Loops and conditional statements
- Scope

“A good FORTRAN programmer can write a good FORTRAN program in any programming language”

Introduction

- Many programming languages - C, C++, Java, FORTRAN, C#, Go, Camel, Python, MATLAB...
- All have advantages and disadvantages - what is your objective?
  - Performance
  - Rapid prototyping
  - Portability
- Which to choose?

Some common choices of programming language

- Performance - FORTRAN, C, C++
- Rapid development - Python, MATLAB, R
- Portability - Java
- Which to choose?

Strengths of C++

- Compiled code - capable of high performance comparable with Fortran, C
- Flexible coding styles - Functional, object oriented, high level, low level
- Powerful standard library with many functions, more added with time (BOOST)
- Local scoping of variables (more later)
- Widespread adoption and support - cross platform, industry, academia

Disadvantages of C++

- A powerful and expansive tool - easy to code for coding's sake (over engineering)
- Matrices and arrays are horrible
- High performance code is harder to write (write for the compiler)
- Cryptic debugging for advanced features, and some not so advanced features

What about C?

- Isn't C++ not just C with extra stuff?
- NOT the same language!
- Relies heavily on pointers to do things (pointers are evil, see later)
- Object orientation is 'roll your own' - bug prone and cumbersome
- A purely 'low level' language
- Archaic and no place in most software (only extremely performance and *memory* limited applications - not very common today)

C++

Type

the ‘type’ of object, e.g. logical, real, integer

Variable

a named object which can store a single value

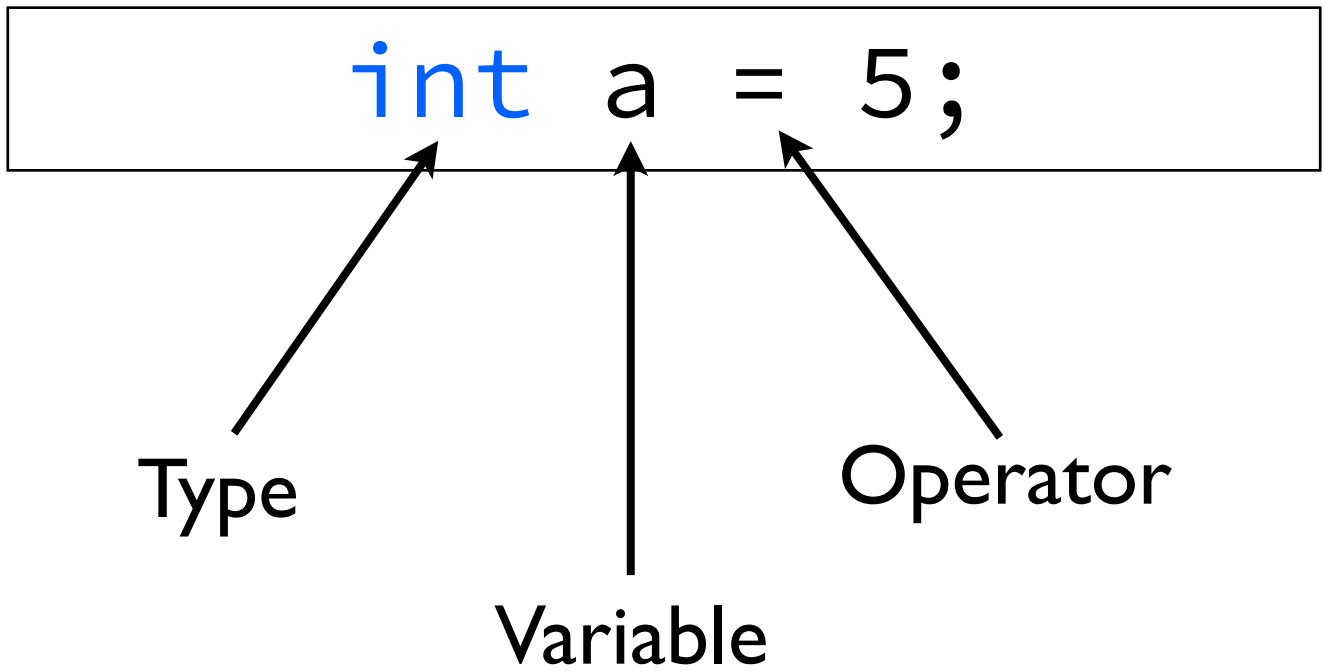
e.g. *a*, *b*, *time*, *distance*...

Operator

something which ‘operates’ on a variable

e.g. +, -, *, /, = ...

Basic statement



Syntax of C++

- C++ code is case sensitive: INT is not the same as int is not the same as Int
- Code is usually written in files with the .cpp file extension
- // signify comments and are ignored by the compiler

Hello World

```
#include <iostream>

int main(){

  std::cout << “hello world” << std::endl;

  return 0;

}
```

include statement
type
code
string

Hello World

Include files/libraries

iostream is part of the C++ standard library and allows input/output to screen

main() is a function of type int and is where all C++ programs start

```
#include <iostream>

int main(){

   std::cout << "hello world" << std::endl;

   return 0;

}
```

return statement to 'return' or end the program
(the function is of type int and so '0' is returned, indicating success)

Hello World

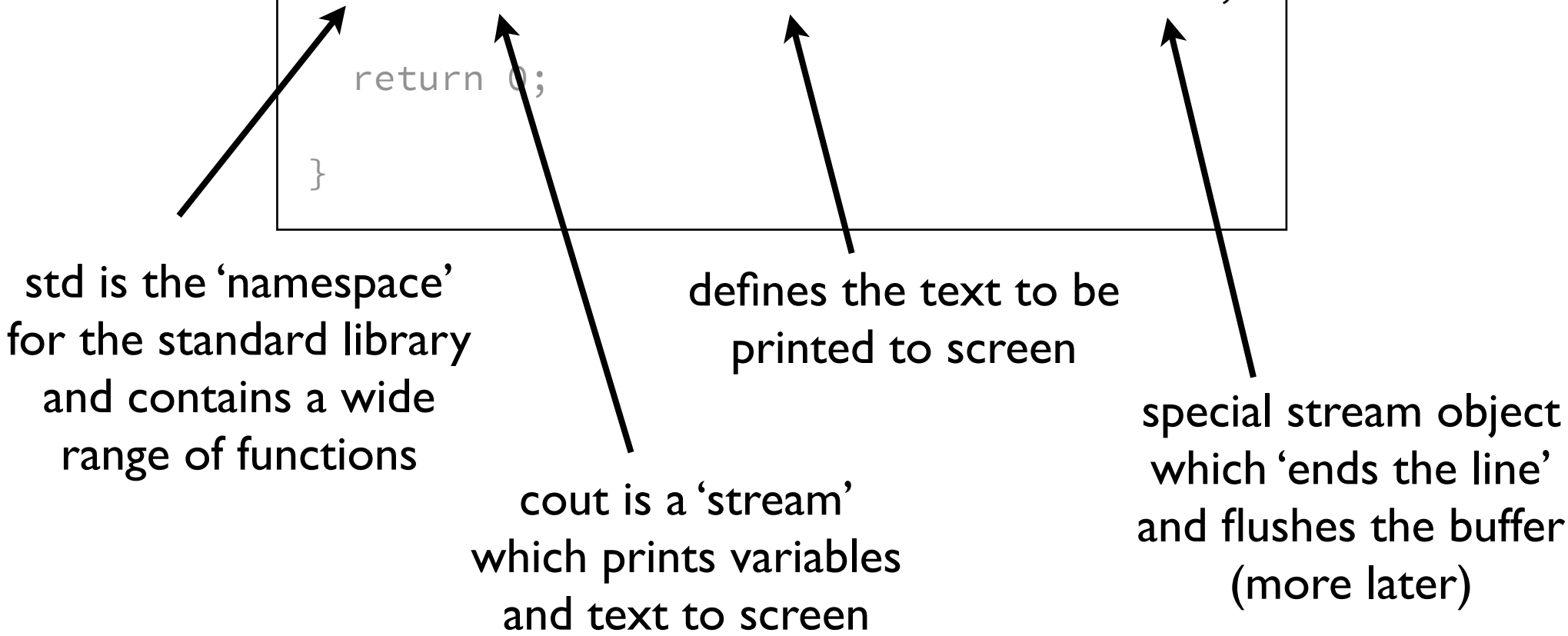
```
#include <iostream>

int main(){

  std::cout << “hello world” << std::endl;

  return 0;

}
```



Hello World

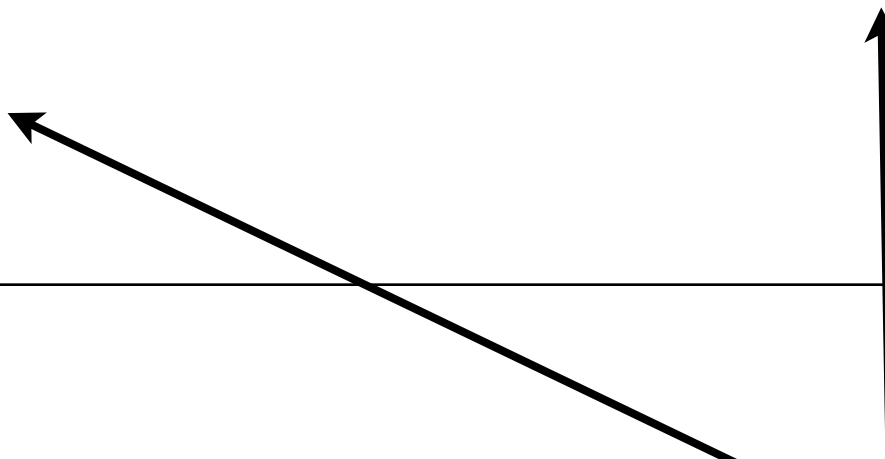
```
#include <iostream>

int main(){

  std::cout << “hello world” << std::endl;

  return 0;

}
```



Almost forgot - semi colon to end each statement; after this course will be automatic;

Variables and Operators

Standard types in C++

```
int       integer 0, 1, 2, 100
bool      logical true, false
float     single precision real number 1.234f, -3.86f
double    double precision real number 2.3456, 1.0e234
char      character variable a,b,c,£,ü etc
```

Let's *declare* an integer variable called 'a'

```
int a;
```

Can also give it an initial value

```
int a = 123;
```

Declaring (initialized) variables

```
int a = 123;
bool flag = true;
float distance = 1.238f;
double time = 1.0;
char character = ‘b’;
```

- Uninitialized variables have a value which is compiler dependent - always initialize your variables
- Real constants are always declared as double precision - use ‘f’ suffix to specify single precision variable

Operators

- An operator ‘operates’ on a variable
- Most basic is the assignment operator

```
a = b;
a = 5;
```

- Assigns value from right to left
- Simple arithmetic operations

```
+   Addition
-   Subtraction
*   Multiplication
/   Division
%   Modulo
```

Operators

- Examples

```
int a = 5;
int b = 3;
int c = a; // c = 5
b = a*c; // b = 25
c = b/2; // c = 12 (truncation)
a = 7;
c = a%2; // c = 1 (remainder 7 - 2*(7/2))
```

- Be careful of order of evaluation

```
b = a*4+c+b-1/3;
b = a*(4+c) + ((b-1)/3); // use brackets
```

Logical Operators

- Logical operators evaluate to true or false

```
==    Equal to
!=    Not equal to
<     Less than
>     Greater than
<=    Less than or equal to
>=    Greater than or equal to
```

- Be careful with real numbers - generally unsafe!
- Also AND, OR and NOT operators

```
&&    Logical AND
||    Logical OR
!     Logical NOT
```

Compound Operators

- Combine arithmetic with assignment

```
a += b          a = a+b
a -= 5          a = a-5
a *= 2          a = a*2
a /= 3          a = a/3
```

- Increment operators

```
++a             a=a+1
--a             a=a-1
```

The C++ Standard Library

Standard Library

- A range of higher level functions and data structures to simplify code development
- Includes strings, mathematical functions, input and output, arrays, lists
- C++ is a minimal language - have to explicitly include library features using include statement:

```
#include <iostream>
```

Input/Output to screen

```
#include <iostream>

// print to screen
std::cout << “Hello: ” << a << std::endl;

// read a from screen
std::cin >> a;
```

- Declares std::cout, std::cin, std:endl and stream operators << and >>
- More in Lecture 3

Common functions

```
#include <iostream>     // Output to screen
#include <cmath>        // math functions
#include <vector>       // vector container
#include <string>       // text strings
#include <fstream>      // output to file
#include <sstream>      // output to string(!)
```

- More information as we go along
- Just remember that you need to include the right component for the part of the library you want to use

Controlling program flow:

Loops and conditional statements

Controlling program flow

- Computers excel at something humans are not good at: doing boring things lots of times
- Programming is the task of telling the computer what you want it to do
- This is done through loops and logical statements
- Loops repeat tasks within the loop multiple times
- logical statements choose which task to do based on a logical (yes/no) choice

The *for* loop

```
for(int i=0; i<10; ++i){
  // any code here is executed 10 times
}
```

- *for* loop is *the* basic loop structure in C++
- Declares a loop variable *i* which only exists inside the loop and controls the number of times the code inside is executed
- *i* = 0 is the initial value and the code is executed as long as *i*<10
- ++*i* increments *i* by one each time the code is executed

for loop examples

```
for(int i=0; i<10; ++i){
  // output i to screen
  std::cout << i << std::endl;
}
```

```
for(int i=0; i<100; ++i){
  // Write my lines to screen 100 times
  std::cout << “I will not throw pencils in class” << std::endl;
}
```

```
for(int i=2; i<13; i+=2){
  // output even numbers up to 12 to screen
  std::cout << i << std::endl;
}
```

for loop examples

```
for(int i=0; i<10; ++i){
  // nested loop (a loop of a loop)
  for(int j=0; j<10; ++j){
    std::cout << i << “\t” << j << std::endl;
  }
}
```

```
// Single line loop without {}
for(int i=0; i<100; ++i) std::cout << i << std::endl;
```

The *if* statement

```
if(a == b){
  // this code is only executed if a == b
}
```

- *if* statement is the basic logical statement
- Checks a condition (e.g. a==b, a<b) that is either true or false and executes the code within only if the condition is true

if statement examples

```
int a = 5;
int b = 7;

// check whether a is less than b
if(a<b){
  std::cout << “a is less than b” << std::endl;
}
```

```
bool today_is_tuesday = true;

// check whether today is tuesday
if(today_is_tuesday==true){
  std::cout << “Go to C++ class” << std::endl;
}
```

if statement examples

```
bool today_is_pancake_day = true;
bool i_have_flour = false;

// Do I need to buy flour?
if(today_is_pancake_day == true && i_have_flour == false){
  std::cout << “Go to buy flour” << std::endl;
}
```

```
bool it_is_raining = true;

// Do I need an umbrella?
if(it_is_raining){ // note: == true not necessary
  std::cout << “Take a Brolly” << std::endl;
}
```

else statements

```
int a = 5;
int b = 7;

// check whether a is less than b
if(a<b){
  std::cout << “a is less than b” << std::endl;
}
// only executed if condition is false (a >= b)
else{
  std::cout << “a is not less than b” << std::endl;
}
```

else statements

```
int a = 5;
int b = 7;

// single line version
// check whether a is less than b
if(a<b) std::cout << “a is less than b” << std::endl;
else std::cout << “a is not less than b” << std::endl;
```

else if statements

```
int a = 5;
int b = 7;

// check whether a is less than b
if(a<b){
  std::cout << “a is less than b” << std::endl;
}
// only executed if condition is false (a >= b)
else if(a==b){
  std::cout << “a is equal to b” << std::endl;
}
else{
  std::cout << “a is greater than b” << std::endl;
}
```

The *while* statement

```
int i = 0;
while(i < 10){
  // this code is only executed if i < 10
  std::cout << i << std::endl;
  // Don't forget to update i each time!
  ++i;
}
```

- *while* statement is a combination of the *for* and *if* statements, which repeats the task as long as the condition is true

The *while* statement

```
int i = 0;
while(i < 10){
  // this code is only executed if i < 10
  std::cout << i << std::endl;
  // I forgot to update i each time
  // Infinite loop as i=0 every time
}
```

Scope

Variable scope

- Defines where a variable is visible in a program
- Important and powerful concept
- Declare variables as you need them - not at the top of functions of the program

Simple example

```
#include <iostream>

int a=2; // visible everywhere -
         // a ‘global variable’

int main(){

  int b=5; // only visible in main()

  std::cout << a << “\t” << b << std::endl;

  return 0;

}
```

```cpp
#include <iostream>

int a=2; // visible everywhere -
         // a ‘global variable’ (bad)

int main(){

  int b=5; // only visible in main()

  // print out a+b 10 times
  for(int i=0; i<10; ++i){
     // declare c inside loop
     int c = a+b;
     std::cout << c << std::endl;
  }

  a = c; // error here - c is not visible
         // outside loop

  return 0;

}
```

Scoping with curly braces

```
#include <iostream>

int main(){

  {
    int b=5; // only visible here
  }
  std::cout << b << std::endl; // error

  return 0;

}
```


Coffee Time