

## Department of Engineering

### 1A Computing Michaelmas Term

# C++

## 1A C++ coursework, Michaelmas Term


This document makes no attempt to exhaustively cover C++ - the recommended book by Deitel and Deitel is 1300 pages long and even that is incomplete.

*Part I* introduces just enough C++ for you to write small programs and run them. The idea is to give you the confidence to learn more yourself - like learning to snowplough as the first stage when skiing, or learning not to be scared of the water when learning to swim. Once you can compile and run programs, you can copy example source code and experiment with it to learn how it works. Like skiing and swimming, you can only learn programming by doing it.

*Part II* introduces aspects of C++ that you could do without when writing this term's code, but they make your programs shorter and easier to read. They're also used in other people's programs. These extra techniques are useful when writing longer programs and are needed for next term.

The C++ language and the methodologies described here will appear strange if you have never written a computer program. Don't attempt to "understand" everything. More explanation will be given later here or in the [CUED Tutorial Guide to C++ Programming](http://www-h.eng.cam.ac.uk/help/languages/C++/c++_tutorial/) ([http://www-h.eng.cam.ac.uk/help/languages/C++/c++\\_tutorial/](http://www-h.eng.cam.ac.uk/help/languages/C++/c++_tutorial/)). For now, concentrate on learning how to use the language. The document includes some quick tests so that you can check your understanding, and some sections of extra information that you can show or hide. If you want to see all the extra sections, click on [this Extras button](#) (<index.php?mode=advanced>). Use **Ctrl +** and **Ctrl -** to adjust the text size.

Work through the document at your own speed. When you've finished exercises 1-4 get a demonstrator to mark your work (6 marks). Try to get at least that far on your first day of programming. When you've had exercises 1-4 marked, continue with the other exercises. When you've finished exercises 5-8, get them marked. If you finish early, you're strongly advised to try some [More exercises](#) ([#Moreexercises](#)) from *Part II*.

Start sessions in the DPO by clicking on the  icon at the bottom of the


screen and then clicking on the **Start IAComputing** icon. This will put some icons on the screen for you, and give you a folder called **1AC++Examples** full of example source code. It also creates a folder called **1AComputing**, a good place to store any course-related files.

If you want to work from home, see our [Installing C++ compilers](http://www-h.eng.cam.ac.uk/help/tpl/languages/C++/InstallingC++compilers.html) (<http://www-h.eng.cam.ac.uk/help/tpl/languages/C++/InstallingC++compilers.html>) page.

In the course of this work your screen might become rather cluttered. The [Window management](http://www-h.eng.cam.ac.uk/help/tpl/new_user) ([http://www-h.eng.cam.ac.uk/help/tpl/new\\_user](http://www-h.eng.cam.ac.uk/help/tpl/new_user)

### Contents

#### Part I

- [Variables](#) ([#Variables](#))
- [Strings and Characters](#) ([#StringsandCharacters](#))
- [Output](#) ([#Output](#))
- [Putting it all together](#) ([#Puttingitaltogether](#))
- [Errors and Warnings](#) ([#Errors](#))
- [Input](#) ([#Input](#))
- [Exercise 1 - Adding](#) ([#Exercise1](#))
- [Decisions](#) ([#Decisions](#))
- [Arithmetic](#) ([#Arithmetic](#))
- [While Loops](#) ([#WhileLoops](#))
- [Exercise 2 - Times Table](#) ([#Exercise2](#))
- [Functions](#) ([#Functions](#))
- [Self-test 1](#) ([#SelfTest1](#))
- [Exercise 3 - Functions](#) ([#Exercise3](#))
- [Arrays](#) ([#Arrays](#))
- [Self-test 2](#) ([#SelfTest2](#))
- [Reading from files](#) ([#Readingfromfiles](#))
- [More about functions](#) ([#Morefunctions](#))
- [Troubleshooting - a checklist](#) ([#Troubleshooting](#))
- [Exercise 4 - Code Solving](#) ([#Exercise4](#))
- [Exercise 5 - Word lengths](#) ([#Exercise5](#))
- [Standard functions](#) ([#Standardfunctions](#))
- [Exercise 6 - Pi](#) ([#Exercise6](#))
- [Exercise 7 - Monopoly](#)  ([#Exercise7](#))

#### Part II

- [Call by Reference](#) ([#CallbyReference](#))
- [Alternative notations](#) ([#MoreNotations](#))
- [More about Loops](#) ([#MoreLoops](#))
- [More about Arrays](#) ([#MoreArrays](#))
- [More Decisions](#) ([#MoreDecisions](#))
- [More types and mixing types](#) ([#MoreTypes](#))
- [Bits, bytes and floatS](#) ([#BitsBytes](#))
- [Enumerations](#) ([#Enumerations](#))
- [Writing to files](#) ([#Writingtofiles](#))
- [Semi-colons](#) ([#Semicolons](#))
- [Exercise 8 - Measuring program speed](#) ([#Exercise8](#))
- [More exercises](#) ([#Moreexercises](#))
- [Useful Links](#) ([#UsefulLinks](#))
- [1A C++ CUED Crib](#) ([crib.php](#))

/desktop/index.html#Windowmanagement) section of the New User Guide has some useful tips.

## Variables [\[ back to contents \(#Contents\) \]](#)

Variables are places to store things. The line

```
int num;
```

creates a variable called `num` in which an `integer` can be stored. Note the final semi-colon - in C++, semi-colons are a little like full-stops at the end of English sentences. You can also have variables that store a

- `float` ("floating point" number - a real number)
- `char` (character)
- `string` (text)

etc. To set an existing variable to a value, use an `=` sign. E.g.

```
num=5;
```

or

```
num=num+1;
```

The latter line might look a bit strange at first sight, but it isn't saying that the LHS is the same as the RHS. It's an *assignment* - it's setting the LHS to the value of the RHS - i.e. adding 1 to `num`.

You can create and set a variable in one line. E.g.

```
float num=5.1;
```

C++ is fussy about variable names - they can't have spaces or dots in them, nor can they begin with a digit. It distinguishes between upper and lower case characters - `num` and `Num` are different variables. It's also fussy about the type of the variable - you can't put text into an integer variable, for example.

[\[show extra information\]](#) (index.php?reply=extravariables#Variables)

## Strings and Characters [\[ back to contents \(#Contents\) \]](#)

Strings are sequences of characters. If you add 2 strings using `+`, the result will be that the 2nd string is appended to the 1st. `strings` are more sophisticated than simple data like `ints` and `floats` (technically speaking, a `string` is an *object*). They have extra functionality associated with them. For example, if you want to find the length of a string `s`, you can use `s.length()`. Here's an example of appending to, then finding the length of, a string

```
string s="hello";  
s=s+" world";  
int l=s.length();
```

To find a particular character in a string (the 3rd, for example) use this method

```
string s="hello";  
char thirdCharacter=s[2];
```

Note that the numbering of the characters starts at 0.

Whereas strings have double quotes around them, characters have single quotes, so to create a character variable and set it to `x` you need to do

```
char c='x';
```

[\[show extra information\]](#) (index.php?reply=extraStringsandCharacters#StringsandCharacters)

## Output [\[ back to contents \(#Contents\) \]](#)

Use `cout` (short for "console output") to print to the screen. Before each thing you print out you need to have `<<`. So

```
cout << 5;
```

prints out a 5, and

```
cout << num;
```

prints out the value of the `num` variable. If you want to print text out, put double-quotes around it - e.g.

```
cout << "hello";
```

prints `hello`. To end a line and start a new one, use the special symbol `endl`. You can print several things out at once, so

```
int num=5;
cout << "The value of num="<<num<< endl;
```

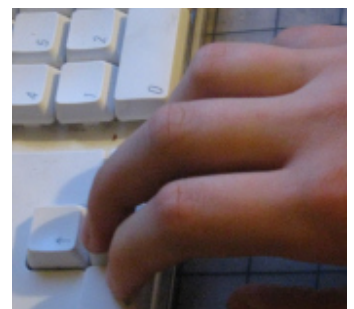
prints

```
The value of num=5
```

## Putting it all together [\[ back to contents \(#Contents\) \]](#)

All the examples so far have been fragments. Now you're going to write complete programs. All the C++ programs that you're likely to write will need the following framework. The Input/Output and string functionality is not actually part of the core language. To use it you need the following lines at the start of your code.

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



In C++ a `function` is a runnable bit of code that has a name. The code might calculate a value (like a function in mathematics does) but it might just perform a task (like printing something to the screen). Every C++ program has a function called `main`. When the program is started, the `main` function is run first. So your program needs a `main` function which will look like this.

```
int main() {
```

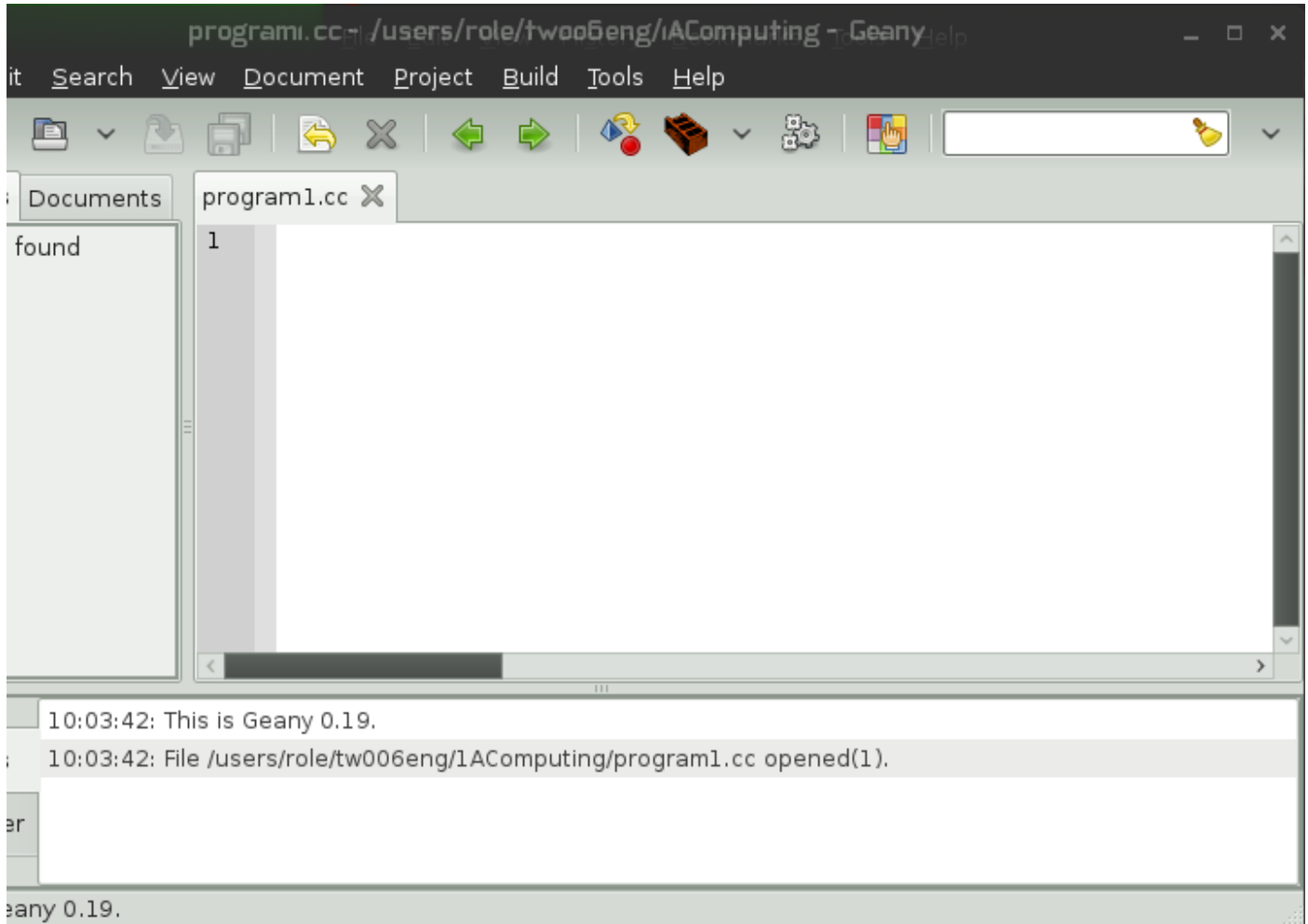
```
    ...
}
```

Don't worry for now what this all means. Just remember that all your programs will probably need lines like these.

Now we'll write a minimal program. At the moment your **1AComputing** folder is nearly empty. In the **File** menu of the **1AComputing - File Browser** window pick the **Create Document** option and create a file called **program1.cc**. Drop the file into the Geany icon





on the desktop and you'll get the following window



**geany** has lots of features to help with writing C++ programs, including an editor. Type (or copy-paste) the following text into it. It's a complete program with some initial setting-up lines and a **main** function containing some code. Note that **//** and anything after it on a line is ignored by the compiler - you can add comments to your programs this way to remind yourself about how they work.

```
#include <iostream> // We want to use input/output functions
#include <string>   // We want to use strings
using namespace std; // We want to use the standard versions of
                    // the above functions.

// My first program! This is the main function
int main() {
    int i=3; // Create an integer variable and set it to 3
    // print the value of i and end the line
    cout << "i has the value " << i << endl;
}
```

In the **Build** menu, choose the **Build** option (or use the  icon). This will try to compile your code. If you've made no typing mistakes then you'll see "Compilation finished successfully" and a file called `program1` will be created, which is in a form that the computer's chip can understand. You'll be able to click on the  icon to run this program. If it prints out `i has the value 3` you've produced your first program!


Note that `geany`

- colour-codes the program to make it more readable. If you don't get colours it's because you haven't named the file with a ".cc" suffix - `geany` expects C++ filenames to have that suffix
- shows line numbers **but those line numbers aren't in the C++ source file.**
- saves the file automatically whenever you do a **build**

[\[show extra information\]](#) (index.php?reply=extraPuttingitaltogether#Puttingitaltogether)

## Errors and Warnings [\[ back to contents \(#Contents\) \]](#)

You may not get everything right first time. Don't be worried by the number of error messages - the compiler is trying to give as much help as it can. You can get a lot of error messages from one mistake so just look at the first error message. Clicking on the error message in `geany` will move the editor's cursor to the corresponding line. Often, the compiler will report that the error is in a line just **after** where the mistake is. If you cannot spot the mistake straight away, look at the lines immediately preceding the reported line.

Testing can only prove the existence of bugs, not their absence - Dijkstra 

The most common errors at this stage will be due to undeclared variables or variables incorrectly declared, and missing or incorrect punctuation. Check to see that brackets match up, and check your spelling. For example, if you have a source file called `program2.cc` with

```
ant i;
```

instead of

```
int i;
```

on line 3 our compiler might give the message

```
program2.cc:3: error: ant does not name a type
```

This message tells you

- the filename (`program2.cc`)
- the line number where the compiler had trouble (line 3)
- a description of the problem.

It may not tell you exactly what's wrong, but it's a clue. Sometimes the compiler doesn't give very helpful messages at all. E.g. if you write

```
cin >> endl;
```

instead of

```
cout << endl;
```

the compiler will give you a page of obscure messages like this

```
/usr/include/c++/4.3/bits/istream.tcc:858: note: std::basic_istream<_CharT, _Traits>&
std::operator>>(std::basic_istream<_CharT, _Traits>&, _CharT&)
[with _CharT = char, _Traits = std::char_traits<char>]
```

Don't panic. All you can do is look at the first line number that's mentioned in the list of errors, and study that line of code.

When you think you've identified the trouble, correct it, save the file and build again.

Even if your code is legal and builds without error, your code may do the wrong thing - perhaps because you've put a '+' instead of a '.'. One of the most effective things to do in this situation is to use `cout` to print out the values of certain variables to help you diagnose where the problem is. Don't just passively stare at your code - make it print out clues for you.

Many common bugs are explained on our [C++ Frequently Asked Questions](http://www-h.eng.cam.ac.uk/help/tpl/languages/C++/FAQ.html) (<http://www-h.eng.cam.ac.uk/help/tpl/languages/C++/FAQ.html>) page. Look there first before asking a demonstrator for help. Many more tips are in the [Troubleshooting](#) (#Troubleshooting) section.

Sometimes compilers will warn you about something that's legal but suspicious. It's worth worrying about such warnings, though they might not be a problem. For instance, if you create a variable called `cnew` and don't use it, the compiler might report

```
warning: unused variable 'cnew'
```

[\[show extra information\]](#) ([index.php?reply=extraerrors#Errors](#))

## Input [\[ back to contents \(#Contents\) \]](#)

Use `cin` (short for "console input") to get values from the keyboard into variables. Before each thing you input you need to have `>>`. The variables need to be created beforehand. E.g.

```
int num;
cin >> num;
```

will wait for the user to type something (they have to press the Return key after). If they type an integer, it will be stored in the `num` variable.

You can input several things on one line, like so

```
int num, angle, weight;
cin >> num >> angle >> weight;
```

[\[show extra information\]](#) ([index.php?reply=extrainput#Input](#))

## Exercise 1 - Adding [\[ back to contents \(#Contents\) \]](#)

You now know enough to write your own programs. Use [geany](#)'s "New" option to create a new file. Save it as `adding.cc` in your [1AComputing](#) folder. You're going to write a program that prints the sum of 2 integers typed in by the user.

I suggest you start by taking a copy of `program1.cc` and removing the contents of the `main` function. Your function needs to

- Create 2 integer variables
- Ask the user to type in 2 integers
- Use `cin` to read the values into your variables
- Print an output line looking rather like this

```
The sum of 6 and 73 is 79
```

Write the code to do this now.

## Decisions [\[ back to contents \(#Contents\) \]](#)

Use the `if` keyword. Here's an example

```
if(num<5) {
    cout << "num is less than 5" << endl;
}
```

The curly brackets are used to show which code is run if the condition is true. You can have many lines of code within the curly brackets. Instead of using `<` (meaning 'less than') to compare values you can use

- `<=` (meaning 'is less than or equal to')
- `>` (meaning 'is greater than')
- `>=` (meaning 'is greater than or equal to')
- `!=` (meaning 'isn't equal to')
- `==` (meaning 'is equal to').

A common error in C++ is to use `=` to check for equality. If you do this on our system with `geany`, the compiler will say

```
warning: suggest parentheses around assignment used as truth value
```

but many compilers won't say anything. Train yourself to use `==` when making comparisons (some languages use 3 equals signs so count your blessings). Be careful to avoid mixed-type comparisons - if you compare a floating point number with an integer the equality tests may not work as expected.

You can use `else` in combination with `if` - e.g.

```
if(num<5) {
    cout << "num is less than 5" << endl;
}
else {
    cout << "num is greater than or equal to 5" << endl;
}
```

You can combine comparisons using boolean logic. Suppose you want to run a line of code if `num` is between 3 and 5. The following **doesn't** work as expected though the compiler won't complain!

```
if (3<num<5)
    ...
```

Instead you need to use

```
if (3<num and num<5)
    ...
```

As well as `and`, the C++ language understands `or` and `not`.

**Don't** put a semi-colon straight after `if(...)`

[\[show extra information\]](#) (index.php?reply=extraDecisions#Decisions)

## Arithmetic [\[ back to contents \(#Contents\) \]](#)

Use `+` (add), `-` (subtract), `*` (multiply), `/` (divide), and `%` (modulus - i.e. getting the remainder in integer division). Note that there's no operator for exponentiation (in particular, `3^2` doesn't produce `9`). The order of execution of mathematical operations is governed by rules of precedence similar to those of algebraic expressions. Parentheses are always evaluated first, followed by multiplication, division and modulus operations. Addition and subtraction are last. The best thing, however, is to use parentheses (brackets) instead of trying to remember the rules.

You can't omit `*` the way you can with algebra. For example you have to use `2*y` to find out what 2 times `y` is. `2y` is illegal.

Although addition, subtraction and multiplication are the same for both `integers` and `floats`, division is different. If you write

```
float a=13.0, b=4.0, result;
result = a/b;
```

then real division is performed and `result` becomes 3.25. You get a different result if the operands are defined as integers:

```
int a=13,b=4;
float result;
result = a/b;
```

`result` is assigned the integer value 3 because in C++, arithmetic performed purely with integers produces an integer as output. If at least one of the numbers is a real, the result will be a real. This explains why later in this document you'll sometimes see `2.0` being used instead of `2` - I want to force real division to be done.

[\[show extra information\]](#) (index.php?reply=extraArithmetic#Arithmetic)

## While Loops [\[ back to contents \(#Contents\) \]](#)

For repetitive tasks, use loops. Easiest is the `while` loop - code that repeatedly runs *while* some condition is true. Here's an example

```
int num=1;
while (num<11) {
    cout << num << endl;
    num=num+1;
}
```

When the computer runs this particular `while` loop, it continues printing `num` and adding one to it while `num` is less than 11, so it prints the integers from 1 to 10.

The indentation of the lines isn't necessary, but it makes the code easier for humans to read, and helps you match up opening and closing braces.

When you write a loop, always make sure that it will eventually stop cycling round, otherwise your program might appear to "freeze". Without the `num=num+1` line in this example, `num` would always be 1 and the loop would cycle forever. If your program does get stuck in

a loop like this, use `geany's`



button to kill the program.

**Don't** put a semi-colon straight after `while(...)`

[\[show extra information\]](#) (index.php?reply=extraWhileLoops#WhileLoops)

## Exercise 2 - Times Table [\[ back to contents \(#Contents\) \]](#)

Use `geany's` "New" option to create a new file. Save it as `timestable.cc` in your `1AComputing` folder. You're going to use a `while` loop to print out the first 10 entries in the 7 times table - i.e.

```

1x7=7
2x7=14
...
10x7=70

```

If you put the `while` loop example inside a `main` routine like the one above, your program will nearly be finished - all you need to do is change the `cout` line so that it not only prints the variable that goes from 1 to 10, but the rest of the line too. Some of the rest of the line doesn't change. The final number does, but it can be expressed in terms of the first number on the line.

## Functions [\[ back to contents \(#Contents\) \]](#)

As we mentioned earlier, in C++ a `function` is a runnable bit of code that has a name. Most programs have many functions. Now you're going to write your own ones.

First we'll re-write exercise 2 using functions. Here's a function called `timesBy7` that multiplies its input by 7.

```

int timesBy7(int number) {
    return number*7;
}

```

Conceptually, C++ functions are rather like maths functions. You can think of them as black-boxes generating output from their input. Note that in C++, functions "return" their output back to the thing that asked for them. Execution of the function code ends when a `return` statement is reached.



The first line of the routine is compact and contains a lot of information.

- `int` - this is saying that the function is going to calculate an integer value rather than (say) a `string`.
- `timesBy7` - this is the name of the function
- `(int number)` - this is saying that `timesBy7` needs to be given an integer as input, and inside this function the integer is going to be known as `number`. Some functions don't need any inputs. Other functions might need many inputs. This function needs exactly one integer.

That first line can be used by itself to "summarise" the function. This line

```

int timesBy7(int number);

```

is called the **prototype** of the function. The compiler needs to know the prototype before it will compile code that uses the function (that's why you need to `include` files at the top of your program; those files contain the prototypes of standard functions). Here's a little program that uses this function to display some multiples of 7. Though short, it illustrates what you need to do when writing your own functions. From now on, just about all your programs in every computing language you learn will use functions, so study this example carefully

```

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

// This prototype is needed to keep the compiler happy
int timesBy7(int number);

int main() {
    int num=1;
    while (num<11) {
        // the next line runs the function and displays the number it returns
        cout << timesBy7(num) << endl;
    }
}

```

```

    num=num+1;
}
}

// This function multiplies the given number by 7
int timesBy7(int number) {
    return number*7;
}

```

When writing a function you need to

- Write the function code
- Add the function prototype to your program (early on, before you use the function)
- Use the function by "calling" it (i.e. running the code in the function). You call a function by typing its name followed by its inputs in brackets. Even if the function needs no inputs you still need the brackets.

This `timesBy7` function needs to be given an integer as input. In this example its output is immediately printed out. You can call it in other ways - for example, you could save its output into a variable called `answer` by doing something like `int answer=timesBy7(9);` or `int x=9; int answer=timesBy7(x);`.

We're now going to write a program with a function that will tell us whether numbers are even or odd. The program's output will be

```

0 is even
1 is odd
2 is even
3 is odd
...
10 is even

```

In the `main` function below we set `i` to 0, 1, ... 10 checking each time to see if `i` is even. The code uses `if` and `else` which are quite easy to understand, I hope. You can read the "`if`" line as saying "if `i` is even, then do ...". This code fragment assumes that C++ has a function called `is_even` which returns `true` or `false`.

```

int main() {
    int i=0;
    while(i<11) {
        if (is_even(i)) { // Or you could have      if (is_even(i)==true) {
            cout << i << " is even " << endl;
        }
        else {
            cout << i << " is odd " << endl;
        }
        i=i+1;
    }
}

```

Read through the code until you understand it. Don't hesitate to trace your finger along the route the computer takes through the code, keeping a note of what value `i` has.

**Self Test 1** - How many times is the "`i=i+1;`" line reached?

Unfortunately there's no function called `is_even` so we'll have to write it ourselves. We could do it many ways. Here we'll use the `%` operator, which gives us the remainder after integer division. If we do `number%2` and the answer is 0, 2 divides exactly into the number, so the number is even. We want the `is_even` function to give us a true/false answer. In C++ there's a type of variable called `bool` (short for Boolean) which can store such answers. Here's the `is_even` function.

```
bool is_even(int number) {
    if ( (number % 2) == 0) {
        return true;
    }
    else {
        return false;
    }
}
```

The prototype of this function is

```
bool is_even(int number);
```

We now have nearly all the code. Below on the left is the complete program with the `main` and `is_even` functions we've prepared. To the right is an animation showing what happens when the program runs for a few cycles

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

bool is_even(int number); // the funct

int main() {
    int i=0;
    while(i<11) {
        if (is_even(i)) {
            cout << i << " is even " << endl;
        }
        else {
            cout << i << " is odd " << endl;
        }
        i=i+1;
    }
}

bool is_even(int number) {
    if ( (number % 2) == 0) {
        return true;
    }
    else {
        return false;
    }
}
```

```
1: i
   0

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

bool is_even(int number);

int main() {
    i=0;
    while(i<11) {
        if (is_even(i)) {
            cout << i << " is even " << endl;
        }
        else {
            cout << i << " is odd " << endl;
        }
        i=i+1;
    }
}

bool is_even(int number) {
    if ( (number % 2) == 0) {
```

Updating displays...done.

DDD: Execution Window

The animation's green arrow starts in the `main` routine at the stop sign and goes round and round the loop. It jumps to and from the `is_even` function. Inside that routine it sometimes follows the `if` route and sometimes the `else` route, depending on whether the number is even or odd.

Don't worry if you can't keep up with the animation. The important thing to realise is that the code isn't simply run from top of the file to the bottom - some lines are run many times. It's also worth noting that the `i` variable exists only in the `main` function - that's why it keeps appearing and disappearing at the top of the animation. It's described as a **local** variable.

When you create a function, don't give it a name that's already in use. Don't, for example, call it `int` because that's part of the C++ language. Don't call it something general (like `vector`, or `max`) because C++ often uses such names internally.

Function definitions can't be nested. In the example code above, for example, the `main` function has to be finished with a final curly bracket before the `is_even` function can be started.

## Exercise 3 - functions [\[ back to contents \(#Contents\) \]](#)

Adapt the previous example so that instead of identifying even numbers it identifies multiples of 3. Give the function a sensible name. Call the program `multiplesof3.cc`.

## Arrays [\[ back to contents \(#Contents\) \]](#)

When you have lots of variables that are related in some way, it's useful to clump them together somehow. Arrays offer a way to do this as long as the variables are all of the same type. For example, if you want to store some integer information about each month in an array, you could use

```
int month[12];
```

to reserve space for 12 integers. Inside the computer the memory layout will be something like this

The integers have the names `month[0]`, `month[1]` ... `month[11]` because array indexing starts at 0. Array items are usually referred to as *elements*. They're stored contiguously in memory. Arrays and loops go well together. To set all the elements of the `month` array to 0, you could do



```
int i=0;
while (i<12) {
    month[i]=0;
    i=i+1;
}
```

This is much shorter than doing

```
month[0]=0;
month[1]=0;
...
month[11]=0;
```

Be careful when using arrays - if you create an array of 5 elements but you use 6 elements, you're using memory that you haven't asked for, memory that might be being used by something else. A crash is likely.

[\[show extra information\]](#) (index.php?reply=extraarrays#Arrays)

## Self-test 2 [\[ back to contents \(#Contents\) \]](#)

Look at this code.

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

int main() {
    int x,y;
    y=3;
```

```
x=y*2;
y=5;

cout << x << endl;
}
```

- How many variables are used?
 

0  1  2  3  4
- When it's compiled and run, what will it print out?
 

2  3  5  6  10
- What is `main`?
 

A variable of type `int`  
 A function that needs 1 integer as input  
 A function that needs integers `x` and `y` as input  
 An array of integers  
 A function that needs no input



## Reading from files [\[ back to contents \(#Contents\) \]](#)

To be able to use the file-reading facilities you need to add

```
#include <fstream>
```

to the top of your file. Then you can do the following to get a line of text from a file (in this case a file called `secretmessage`) and put it into a string (in this case called `message`).

```
string message;
ifstream fin; // a variable used for storing info about a file
fin.open("secretmessage"); // trying to open the file for reading
// The next line checks if the file's been opened successfully
if(fin.good()) {
    // we've managed to open the file. Now we'll read a line
    // from the file into the string
    getline(fin, message);
}
else { // print an error message
    cout << "Couldn't open the secretmessage file." << endl;
    cout << "It needs to be in the same folder as your program" << endl;
    return(1); // In the main function, this line quits
              // from the whole program
}
```

This code includes a check to see if the file exists. There are many other file-reading facilities, but that's all you'll need for now.

[\[show extra information\]](#) (index.php?reply=extraReadingfromfiles#Readingfromfiles)

## More about functions [\[ back to contents \(#Contents\) \]](#)

The functions we've seen so far have 0 or 1 input values and 1 output value, but C++ is more flexible than that.

- Often the purpose of a function is to work out a value and return it, but sometimes a function will perform a task (like printing to the screen) and won't have anything useful to return. In such situations, the function needn't return anything. If a function called `printOut` returns nothing and needs no input, its prototype would be

```
void printOut();
```

where the `void` word means that nothing is returned.

- Functions can have several input values. For example, a function with the prototype

```
float pow(float x, float y);
```

needs to be given 2 floating point numbers. It returns a floating point number too.

The following table shows how C++ represents various types of function diagrams

C++	Function diagram
<pre>int fun1(int x); int i=fun1(7);</pre>	
<pre>void fun2(int x); fun2(7);</pre>	
<pre>int fun3(void); int i=fun3();</pre>	
<pre>int fun(int x); int i=fun(fun(7));</pre>	

The variables created in a function can only be used in that function - they're local variables. If you have "int i;" in your `main` function and another "int i;" in another function the 2 `i` variables will be independent.

Try the "function" [teaching aid](http://www-h.eng.cam.ac.uk/help/tpl/languages/C++/C++function.php) (<http://www-h.eng.cam.ac.uk/help/tpl/languages/C++/C++function.php>) to get some practise

## Troubleshooting - a checklist [\[ back to contents \(#Contents\) \]](#)

The longer your programs become, the harder it will be to fix bugs and the more important it will be to write tidy code. Here are some things to check

- Variables**
  - Are they created at the right time? Do they have the right initial value?
  - Are they the right type? (`int` rather than `float` maybe, or an array instead of a simple value)
  - Are they sensibly named? Are you clear about what each variable's for and what its contents represent?
- Program Organisation**
  - Do you have a `main` function? Does it begin with `int main()`?
  - Have you `#included` the right files? Do you have `using namespace std;`?
  - Do you know when one function ends and another begins?
  - Do all your blocks of code (in `while` loops, `if .. else` constructions, etc) start and end where they're meant to?
  - Have you put any comments in your code?

- *Functions*
  - Do they have the right inputs and outputs?
  - Does the prototype's input/output specification match the function code's specification?
  - Are you calling the function? How do you know? (add `cout` commands). Remember that when you call a function you need brackets after the name even if the function requires no input values.
  - If you're using random numbers are you calling `srandom` exactly once?
- *Arrays and Loops*
  - Do your loops go round forever?
  - Do you go off the end of an array?
  - Did you remember that the 1st item in an array has an index of 0?
- *Punctuation*
  - Are you mixing up `=` and `==`?
  - Are you using semi-colons correctly?
  - `if` and `while` need to be followed by a condition in brackets. Have you added the brackets?
- *Strategies and workflow*
  - If the code's not compiling, make it compile by removing (or commenting-out) problematic lines until it does. Then restore a line or 2 at a time. Re-build after each addition. Print out some variables to see if the code is behaving.
  - If you've changed your code but your program's behaviour hasn't changed, maybe you haven't created a new version of the program. Remember, the `Compile` button doesn't create a new program, but the `Build` menu item does - `F9` is a short-cut for it.
  - When the code runs but does the wrong thing, run through the code on paper as if you were the computer, keeping a note of variables' values.
  - Read through the [C++ Frequently Asked Questions](http://www-h.eng.cam.ac.uk/help/tpl/languages/C++/FAQ.html) (<http://www-h.eng.cam.ac.uk/help/tpl/languages/C++/FAQ.html>) and the [C++ CUED Crib](#) ([crib.php](#)) ([crib.php](#)) ([crib.php](#)) ([crib.php](#))

(crib.php)

## (crib.php) Exercise 4 - Code Solving (reading files) [\[ back to](#)

[contents](#) (#Contents) ]

In your 1AComputing folder there's a file called "secretmessage". Your mission is to write a program called `decode.cc` to decode the contents of this file. It contains a single line of text that's been encoded using a simple shifting algorithm where each letter has been replaced by the letter N places before it in the alphabet (Z being treated as the letter before A). Spaces haven't been encoded, and there's no punctuation. Just try to decode the message, determining the appropriate N by trial and error. The message is in upper-case letters. Break the task into stages, checking as you go

- Using the code in the previous section, read the message in from the file into a `string` using `getline` (the message is only one line long). Display the encoded message on your screen.
- Create an integer `N` (the amount the characters have been shifted by) and set it to a value. For now, let's just guess the number 7.
- You can use the string as if it were an array of characters. Pick out the characters in the string one at a time using a loop like this

```
char c;
// The next variable is declared as unsigned (i.e. non-negative)
// so that the compiler doesn't warn us later
unsigned int characters_processed=0;
// while we're not at the end of the string, get the next character
while(characters_processed < message.length()) {
    c=message[characters_processed];
    characters_processed=characters_processed+1;
}
```

Each character is represented in the computer as an integer, so you're allowed to perform arithmetic on the value.

- Inside the `while` loop, process each character `c`.
  - If the character is a space (i.e. if it's equal to `' '`), print it
  - If the character is not a space, set `c` to be `c` plus `N`. If the resulting character is more than `'Z'` (i.e. `if (c > 'Z')`), subtract 26

from `c` so that the letter cycles round. Print the resulting `c` out.

Build and run the program. If you're lucky you'll get a readable message. It's more likely that you'll get junk because `N` will need to be a different number. Rather than manually having guess after guess (changing `N`, recompiling and running until you get the answer) try this

- Restructure the code to make it neater. This is something developers often have to do - re-engineer their work so that it does the same as before, but in a tidier way. Rather than have one big function you're going to create 2 shorter functions. Put the decoding code you've written into a function called `decode_and_print` that takes the coded message and `N` as its input parameters, returning nothing. Its job is to print the decoded message. What will its prototype be? Once you've written the function and added its prototype to the file, all that your `main` function need contain is the code to read the message in, then a call to the function. You should get the same result as before!
- Now change the `main` routine so that it tries decoding with `N=1`, then `N=2` up to `N=25`. printing the decoded message each time. Do this using a loop. Determine the `N` that works for your message.

When you've completed programs 1 to 4 get them marked. Make sure that you have restructured the code - a single 40-line `main` function isn't good enough.

## Exercise 5 - Word lengths [\[ back to contents \(#Contents\) \]](#)

Your next task is to read the words in a file, find their lengths and print the frequency of word-lengths from 1 to 10. Call the program `wordlengths.cc`

### Tips

- Create a file with some words in it - just a few words initially, several per line if you want. This file needs to be in the same folder as your program. Remember to save the file.
- Read the words one by one into a string using code like this

```
#include <fstream> // so that file-reading works

string word;
ifstream fileInput; // a variable of a type that lets you input from files
fileInput.open("filename"); // this tries to open a file called "filename"
// The next line uses 'not', a keyword
if(not fileInput.good()) { // print an error message
    cout << "Couldn't open the file." << endl;
    cout << "It needs to be in the same folder as your program" << endl;
    return(1);
}

// the next line of code reads a word from a file and puts it into the
// variable 'word'. It does so while there are words left in the file
while(fileInput >> word) {
    // print it out to check that the code's working
    cout << word << endl;
}
```

Check that this works as expected before going on to the next stage. You'll need to write a `main` function to contain this code, and `include` some files at the top.

If you want to know how this works in more detail, read a book.

- Create an array of variables called `frequency` to store the frequencies. Easiest is to arrange things so that `frequency[1]` is used to store the number of words of length 1, `frequency[2]` is used to store the number of words of length 2, etc. Initialise these variables to 0. Change your code in the `while` loop so that 1 is added to the appropriate variable each time you've read in a word and found its length. For example, if the length of a word is 3 you'll need to add 1 to `frequency[3]`. More generally, if the word length is `len`, you need to add 1 to `frequency[len]`. Once all the words have been read, this array of variables will hold the final frequency counts.
- Print a simple table of the results like the following, using a loop

Length	Frequency
1	3
2	6
...	
10	1

- When you have this working with a little file of words, try it with a bigger file - you could even try downloading something from [Project Gutenberg](http://www.gutenberg.org) (<http://www.gutenberg.org>). What are you going to do if `len` is more than the number of elements in the `frequency` array?

If you haven't got the hang of using arrays to store how many times things happen, read the [I don't understand how count things and store the frequencies in an array](http://www-h.eng.cam.ac.uk/help/tpl/languages/C++/FAQ.html#counting) (<http://www-h.eng.cam.ac.uk/help/tpl/languages/C++/FAQ.html#counting>) item.

## Standard functions [\[ back to contents \(#Contents\) \]](#)

By putting extra lines at the top of your file you gain access to further functions that have already been written. Here are some examples.

- `#include <cmath>`

With this you can use maths routines - `sin(x)` (which uses radians); `pow(x,y)` (which raises `x` to the power `y`), `sqrt(x)`, etc.

- `#include <cstdlib>`

With this you can access functions that generate random numbers. Before the random number generator is used for the first time, call

```
srandom(time(0));
```

so that you get a different sequence of random numbers each time you run the program. Don't call it more than once. After that, each time the function `random()` is called, it will return a random positive integer. Work out what the following function does and how it works.

```
int RollDie()
{
    int randomNumber, die;

    randomNumber = random();
    die = 1 + (randomNumber % 6);
    return die;
}
```

You'll be using this code later, so if you've any doubts about what this code does, write a `main` function that calls it, add some `cout` statements to print useful variables out, then build and execute the code.

The numbers produced by the `random` routine are only pseudo-random. Here we're using the computer's real time clock as the seed. See [wikipedia's Random number generation](http://en.wikipedia.org/wiki/Random_number_generation#Computational_methods) ([http://en.wikipedia.org/wiki/Random\\_number\\_generation#Computational\\_methods](http://en.wikipedia.org/wiki/Random_number_generation#Computational_methods)) page if you want more details. For more information about functions in general, see

- A [section of the C++ tutorial guide](http://www-h.eng.cam.ac.uk/help/languages/C++/c++_tutorial/functions.html) ([http://www-h.eng.cam.ac.uk/help/languages/C++/c++\\_tutorial/functions.html](http://www-h.eng.cam.ac.uk/help/languages/C++/c++_tutorial/functions.html))
- A [more sophisticated animation](http://www-h.eng.cam.ac.uk/help/ahg/1AComputing/animations/power.mc) (<http://www-h.eng.cam.ac.uk/help/ahg/1AComputing/animations/power.mc>) (which might only work in the DPO)
- An answer to a [frequently asked question](http://www-h.eng.cam.ac.uk/help/tpl/languages/C++/FAQ.html#1.22) (<http://www-h.eng.cam.ac.uk/help/tpl/languages/C++/FAQ.html#1.22>)

## Exercise 6 - Pi (math functions) [\[ back to contents \(#Contents\) \]](#)

You can calculate pi just by dropping a pen, as long as you drop it onto floorboards several times. If the pen is as long as the floorboards are wide, then pi will be roughly

$$2.0 * \text{number\_of\_drops} / \text{number\_of\_times\_pen\_lands\_on\_crack}$$


The pen will hit a crack (an edge of a floorboard) if the closest distance ( $D$ ) from the pen's centre to a line is less than or equal to 0.5 times  $\sin(\theta)$ , where  $\theta$  is the angle between the pen and the cracks.



This gives us the chance to do a simulation (a common task in engineering). Don't try to write the complete program before trying to build it. Take it stage by stage, compiling and testing as you go.

1. In a file called `pi.cc` write a function to simulate the dropping of the pen. It could have the prototype

```
bool dropthepen();
```

returning `true` if it touches a crack in the floor and `false` otherwise.

There are 2 random factors to take account of - the angle of the pen (0-90 degrees) and the distance of the pen's centre from a line (0 - 0.5; our floorboards will be 1 unit wide). The function will need to calculate  $\sin(\theta)$ . The C++ `sin` function expects its argument to be in radians. You can get round that by doing

```
float angleindegrees=(90.0*random())/RAND_MAX; // a number between 0 and 90
float angleinradians=angleindegrees*M_PI/180; // a number between 0 and pi/2
```

These expressions use variables made available by the inclusion of `cstdlib` and `cmath`: `RAND_MAX` is the biggest integer that the `random` routine produces, and `M_PI` is the value of  $\pi$ .

Create a variable `D` and set it to  $(0.5 * \text{random}()) / \text{RAND\_MAX}$  (a random number between 0 and 0.5).

Using `D` and  $\sin(\text{angleinradians})$  you can now work out whether the pen lands on a crack.

2. Add a `main` routine that calls your function. Before you call the function, call `srandom(time(0))` once to initialise the random number generator (remember, you'll need `#include <cstdlib>` at the top of the file to access these random number routines). Use a `while` loop to call `dropthepen` 10 times. Does the function seem to be working ok? How many times do you expect the pen to touch a crack? If it touches 0 or 10 times, your code is probably wrong.
3. Before the `while` loop, create a variable (`numberOfHits`, say) to store the number of times the pen crosses a crack. Initialise it appropriately. Inside the `while` loop, add 1 to it when the `dropthepen` function returns `true`. After the `while` loop, write the code to estimate  $\pi$ . If `pi` comes out to 3, make sure you're using `2.0` rather than `2` when calculating (when you divide an `int` by an `int` in C++, you get an `int`. By involving a floating point number in the calculations, floating point arithmetic will be performed).
4. Now do 10,000 runs. You should get a more accurate answer for  $\pi$ .

Your program is likely to have the following layout - some included files, a prototype and 2 functions.

```
#include <iostream>
// other included files

// prototype
bool dropthepen();

// main function
int main {
    // Initialise random number generator, call dropthepen many times,
    // gather statistics and print the answer
}

// dropthepen function
bool dropthepen() {
    // return true if pen lands on crack, otherwise return false
}
```

## Exercise 7 - Monopoly © [\[ back to contents \(#Contents\) \]](#)

You're playing [Monopoly](http://www.guppimedia.com/Monopoly/04Cambridge.html) (http://www.guppimedia.com/Monopoly/04Cambridge.html) . First the good news - you've landed on GO so you get £200. Now the bad news - there are 5 hotels along the first side of the board and you own none of them. What chance do you have of reaching the 2nd side without landing on any of the hotels? What's the most likely number of hotels that you'll land on?



You could solve this using probability theory. We're going to do it "by experiment", writing a program called `monopoly.cc` to run 10,000 simulations.

Whenever you have a non-trivial program to write, think about how it can be broken down into stages, and how you can check each stage.

1. You've already seen the `RollDie` function that simulates the rolling of a single die. Copy it into your new file. Now write a function called `Roll2Dice` to simulate the rolling of 2 dice (call `RollDie` twice and return the sum of the answers). Before going any further, test it. If it doesn't work, neither will your full program! Here's a `main` function you could use to test it

```
int main() {
    srand(time(0));
    cout << "Roll2Dice returns " << Roll2Dice() << endl;
}
```

You'll need to add prototypes for `RollDie` and `Roll2Dice` too.

2. Write a function to simulate a single user's attempt to get past the hotels. Call it `runTheGauntlet` unless you can think of a better name. It won't need any inputs, but it needs to return the number of hotels landed upon. Think first about your strategy in words, then express that strategy using C++. It helps to have a variable (called `location`, say) that records where you are. First set a counter `hotelsVisited` to 0. You need to keep rolling the dice until you've gone past the 9th square (so use a `while` loop). Each time you move, check to see whether you've landed on a hotel. If you have, add 1 to `hotelsVisited`. At the end of the function return `hotelsVisited`. Test this function - run it from the `main` function a few times and print the outcome to see if the results are reasonable.

To see whether you've landed on a hotel, use something like

```
if (location==1 or location==3 or location==6 ...
```

and *not*

```
if (location==1 or 3 or 6 ...
```

The latter isn't illegal but it doesn't do what you might expect.

3. Now call that function 10,000 times using something like

```
int output=runTheGauntlet();
```

in a loop. You don't want to print each outcome but you do want to store how many times no hotels were landed on, how many times only 1 hotel was landed on, etc. Create an array called `frequency` to store the results. `frequency[0]` will contain the number of runs when no hotels were landed upon, so for each run when no hotels are landed on you need to add 1 to this value. When you've done all the runs, `frequency[0]` will contain the final value. You need to deal similarly with the other elements of the array. How many elements will this array need? What should the initial value of each element be?

4. Using a loop, print the summary of results on the screen.

```
Hotels-visited  Frequency  Percentage
0
...
5
```

To print the columns out neatly, these 2 facilities might help

- `setw` - this lets you set the minimum number of characters produced by the next piece of output.
- `setfill` - with this you can choose the character that will fill the gaps caused by using the `setw` command

So if you want to print out an integer `i` to fill a column 10 characters wide, filling the gaps with blanks, you could use

```
#include <iomanip>
...
setfill(' ');
cout << setw(10) << i ;
```

Note: On each turn in Monopoly, 2 dice are thrown and their sum used to determine how far a player's counter moves. If the GO square is counted as position 0, then the hotels are located on positions 1, 3, 6, 8 and 9 along the first side. In the Cambridge Edition of Monopoly, the properties along the first edge are (in order) "Histon Road", "Parker's Piece", "Hills Road", "Cheddars Lane" and "Bateman Street". As an extra challenge you could find out which property you are most likely to land on.



## Part II [\[ back to contents \(#Contents\) \]](#)

See the [CUED Tutorial Guide to C++ Programming](http://www-h.eng.cam.ac.uk/help/languages/C++/c++_tutorial/) for a more formal and complete introduction to the C++ language.

## Call by Reference [\[ back to contents \(#Contents\) \]](#)

Suppose we want to write a function that will triple the value of a given variable. We could try the following

```
#include <iostream>
using namespace std;
// prototype
void triple(int i);

int main()
{
    int i=3;
    cout << "In main, i is " << i << endl;
    triple(i);
    cout << "In main, i is now " << i << endl;
}

void triple(int i) {
    i=i*3;
    cout << "in triple, i becomes " << i << endl;
}
```

But it doesn't work as we wanted - the `i` in `main` doesn't change. Run it if you're not convinced. The problem is that `main`'s `i` is a different variable to the `i` in `triple` - each is a variable that's local to the function it's in. That the 2 variables have the same name is a coincidence (if the `i` variable in the `triple` function were renamed, the code would work just as well). The only link between the 2 variables is that when `triple` is called, `triple`'s `i` gets its initial value from `main`'s `i` - `i` is being passed "by value".

If we want to change `main`'s `i` we need to "call by reference" - note the added ampersands in the following code.

```
#include <iostream>
using namespace std;
// prototype
void triple(int& i); // added ampersand

int main()
```

```

{
int i=3;
    cout << "In main, i is " << i << endl;
    triple(i); // NO added ampersand
    cout << "In main, i is now " << i << endl;
}

void triple(int& i) { // added ampersand
    i=i*3; // here i is an alias for main's i
}

```

When a variable is passed using "call by value", the function is given the variable's value. The function doesn't know what the original variable was, so the function can't change it. When a variable is passed using "call by reference" the function can "refer to" the original variable, so it can be changed.

Why is "call by reference" useful? One reason is that it lets functions "return" more than one value. If, for example, you write a function that has 3 input parameters that are passed "by reference", the function can change all 3 of them, and those changes will be visible outside the function.

## Alternative notations [\[ back to contents \(#Contents\) \]](#)

C++ has alternative ways to do some things

- *Comments* - Instead of using `//` to comment out a line, you can use `/* ... */` to comment out a block of text
- *Increment/decrement* - there are shortcuts to changing a variable's value.

This	is equivalent to	this
<code>i++;</code>	...	<code>i=i+1;</code>
<code>i--;</code>	...	<code>i=i-1;</code>
<code>i+=3;</code>	...	<code>i=i+3;</code>
<code>i-=8;</code>	...	<code>i=i-8;</code>

- After `if`, `while`, etc we've always put the next block of code in curly brackets. If (and only if) the block consists of one statement, the brackets aren't needed. So

```

if(2+2==4) {
    cout << "Correct!" << endl;
}

```

can be written as

```

if(2+2==4)
    cout << "Correct!" << endl;

```

or even

```

if(2+2==4) cout << "Correct!" << endl;

```

## More Loops [\[ back to contents \(#Contents\) \]](#)

In a `while` loop you sometimes want to abort early. There are 2 commands to do this

- `break` - this breaks out of the loop completely
- `continue` - this breaks out of the current cycle and starts the next one.

The best way to understand these commands is to see them in action. If you run this program, what would it print out? If you're not sure, **run it and see!**

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

int main() {
    int i=0;
    while(i<10) {
        i=i+1;
        if(i==2)
            continue;
        if (i==4)
            break;
        cout << "i=" << i << endl;
    }

    cout << "End of looping" << endl;
}
```

Another way to do looping is to use a **for** loop. Earlier we had this **while** loop.

```
int num=1;
while (num<11) {
    cout << num << endl;
    num=num+1;
}
```

Notice that it has

- Initialisation code - **int num=1**
- Code to control termination - **num<11**
- Code that's run each cycle to make the next cycle different - **num=num+1**

With a **for** loop all this code that controls the cycling is brought together in a compact form. Here's the **for** loop equivalent of the above **while** loop

```
for (int num=1; num<11; num=num+1) {
    cout << num << endl;
}
```

or more commonly

```
for (int num=1; num<11; num++) {
    cout << num << endl;
}
```

Note the format -

**for ( *initialisation* ; *decision* ; *each cycle* )**

"for" loops are more common than **while** loops (people like having the "loop-controlling" code together) so you'll have to get used to them. Try the ["for" loop teaching aid](http://www-h.eng.cam.ac.uk/help/tpl/languages/C++/C++forloop.php) (http://www-h.eng.cam.ac.uk/help/tpl/languages/C++/C++forloop.php) to get some practise and try re-writing some of the earlier exercises using **for** loops instead of **while** loops.

C++'s **for** loop is very flexible. The "loop variable" doesn't need to start at 1, nor do you need to add 1 to it each time you go round the loop. For example, the following code prints 11, 10 ... 1.

```
for (int num=11; num>0; num--) {
```

```
    cout << num << endl;
}
```

[\[show extra information\]](#) (index.php?reply=extraMoreLoops#MoreLoops)

## More about Arrays [\[ back to contents \(#Contents\) \]](#)

- An array can be passed to a function as an input parameter. Arrays are always "passed by reference". Here's an example.

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

void timesarrayby2(int numbers[]); // The square brackets are needed
// because 'numbers' is an array. Note that there's no ampersand
int main() {
    int nums[10];
    for(int i=0; i<10; i++) {
        nums[i]=i;
    }
    timesarrayby2(nums);
    // Note that the numbers in the array have changed.
    for(int i=0; i<10; i++) {
        cout << nums[i] << endl;
    }
}

void timesarrayby2(int numbers[]) {
    for(int i=0; i<10; i++) {
        numbers[i]=2*numbers[i];
    }
}
```

- Earlier we used 1-dimensional arrays but you can create arrays with more dimensions. Here's an example of a 2D array that stores 6 integers in 2 rows of 3 columns. If you want to set all of the elements to 0 you could use "nested loops" - loops inside loops. The following code sets `table[0][0]` to zero, then `table[0][1]` to zero, etc

```
for (int row=0; row<2; row++)
    for(int column=0; column<3; column++)
        table[row][column]=0;
```

```
int table[2][3];
```

[0][0]	[0][1]	[0][2]
[1][0]	[1][1]	[1][2]

## More Decisions [\[ back to contents \(#Contents\) \]](#)

- There are older, common alternatives to the boolean operators

This	<i>is equivalent to</i>	this
&&	...	and
	...	or
!	...	not

- You can "nest" ifs. In the following code, the 2nd `if` line is only reached if `num < 5` is true.

```
if (num < 5) {
    cout << "num is less than 5" << endl;
    if(num < 3) {
        cout << "and num is less than 3" << endl;
    }
}
```

```
    }
    else {
        cout << "but num is equal to or greater than 3" << endl;
    }
}
else {
    cout << "num is greater than or equal to 5" << endl;
}
```

Follow the route that the execution of this code takes when `num` is 6, then 4, then 2.

- Sometimes you might want to perform a different action for each of many possible values of an integer variable. You could use `if` a lot of times. Alternatively you can use `switch`. Here's an example

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

int main() {
    int i=0;
    while(i<10) {
        switch (i) {
            case 0: cout << "i is zero" ;
                    break;
            case 1: cout << "i is one" ;
                    break;
            case 2: cout << "i is two" ;
                    break;
            default: cout << "i isn't 0, 1, or 2";
                    break;
        }
        i=i+1;
        cout << endl;
    }
}
```

When `i` is 0, the "case 0" block is run. In this situation the `break` doesn't break out of the surrounding `while` loop, it breaks out of the `switch`. Without the `break`, execution would "fall through" into the `case 1` block. Try to work out what happens in this code before running it. What does it print out?

- If you decide to quit from the program completely, use `exit(1);`. For this to work you need `#include <cstdlib>` at the top of your file.

## More types and mixing types [\[ back to contents \(#Contents\) \]](#)

- We've already mentioned that there are variable type of C++ variables: `int` for integers, `float` for floating point numbers, `char` for characters and `bool` for booleans (`true` or `false`). There are `doubles` (which are floating point numbers too, but potentially more accurate than `floats`) and `longs` (which store integers that might be too big to fit into an `int` variable). You can also declare that variables will only store non-negative values by using the `unsigned` keyword. For example, `unsigned int height;` creates a variable whose contents won't be negative.

C++ is quite strict about types. What does the following program print out?

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

int main() {
    int i=1;
    int j=2;
    cout << "i/j=" << i/j << endl;
}
```

Did you hope it would print out `1/2=0.5`? Actually it prints out `1/2=0` because in C++ an arithmetic operation with only integer operands results in a (possibly rounded-down) integer. If at least one operand is a floating point number, the answer's a floating point number, so you could make this program print out `1/2=0.5` by changing it to

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

int main() {
    int i=1;
    int j=2;
    cout << "i/j=" << i*1.0/j << endl;
}
```

- If the range of types that C++ provides is too restrictive you can invent new types. When you use a language (like English or C++) you are often attempting to represent or model the real world. The more the modelling language structurally resembles what it's modelling, the easier the modelling is likely to be. If for example your program was dealing with 2-dimensional points, it would help to have a type of variable to represent a point. In C++ you can create such a type like this

```
class Point {
public:
    float x;
    float y;
};
```

Note that this **doesn't** create a variable, it creates a new **type** of variable. `Point` is called a **class**. To create a variable of type `Point` you do the same kind of thing that you did when creating variables of type `int`, etc. To create an `int` called `i` you would do

```
int i;
```

To create a `Point` called `p` you do

```
Point p;
```

The following code fragment shows how to set `p`'s component fields to values.

```
p.x=5;
p.y=7;
```

Whereas in arrays all the elements needed to be of the same type and each element was identified by a number, in classes they

can be of different types and are each given a name. Suppose you have the following data

Name	Anna	Ben	Charlie
Height	1.77	1.85	1.70
Age	20	18	15

You can create a class designed to contain this information as follows

```
class Person {
public:
    string name;
    float height;
    int age;
};
```

You could then create a variable to represent Anna's information by doing

```
Person anna;
anna.name="Anna";
anna.height=1.77;
anna.age=20;
```

If you wanted to print Anna's height later on, you could do

```
cout << "Anna's height = " << anna.height << endl;
```

If we have several people, it's useful to create an array. The syntax when creating arrays of new types is the same as when creating arrays of built-in types like `ints`. The following line creates an array called `family` big enough to store information about 3 people;

```
Person family[3];
```

To set the age of the first person to 20 you'd do

```
family[0].age=20;
```

[\[show extra information\]](#) (index.php?reply=extraMoreTypes#MoreTypes)

## Bits, bytes and `floats` [\[ back to contents \(#Contents\) \]](#)

You may sometimes want to access the individual bits of a byte. You'll need to do so when programming robots in the 2nd year, and questions about bits are often in 1st year exams. As you'll see in the lectures, a variable of type `float` is stored in a standard format using 4 bytes, each of 8 bits. Each bit can be off or on (a binary digit - 0 or 1). You might want to write programs to reinforce your theoretical knowledge. First we'll look at bits and bytes. We need to use the `&` (aka `bitand`) operator, which performs a bit-wise `and` with the 2 operands, and the `|` (aka `bitor`) operator, which performs a bit-wise `or`. Look at this program

```
#include <iostream>
using namespace std;
int main() {
    unsigned char num=43;
    unsigned bitmask=128; // the bit pattern 10000000
    for (int thebit=7;thebit>-1;thebit--) {
        // Now see if (num bitand bitmask) is non-zero
        if (num bitand bitmask) {
```

```

        cout << "1";
    }
    else {
        cout << "0";
    }
    bitmask=bitmask/2;
}
cout <<endl;
}

```

It goes through the bits of the byte called `num` checking the value of each bit (most significant first) and printing it out, so that in the end you get a binary representation of the decimal number 43.

If you do `num & x` and `x` is all 1s, then the answer will be the same as `num`. If `x` has exactly 1 bit set to 0, then `num & x` will be the value of `num` with that bit set to 0. So if you wanted to set the 4th bit from the right to 0 in `num` you could do

0	0	1	0	1	0	1	1	43
&								
1	1	1	1	0	1	1	1	247
=								
0	0	1	0	0	0	1	1	35

```
num = num & 247;
```

Similarly, to set the 3rd bit from the right to 1 you could do

0	0	1	0	1	0	1	1	43
0	0	0	0	0	1	0	0	4
=								
0	0	1	0	1	1	1	1	47

```
num = num | 4;
```

Finding the bit pattern of a floating point number requires knowing a little more about C++ than is in the syllabus. The code below uses the code above to display the bits of a floating point number byte by byte. You can use it to revise floating point representation (note that on PCs the most significant byte is at the highest memory address - i.e. it's printed out last by this program)

```

#include <iostream>
using namespace std;

void printbinary(unsigned char num) {
    unsigned bitmask=128;
    for (int thebit=7;thebit>=0;thebit--) {
        if (num & bitmask)
            cout << "1";
        else
            cout << "0";
        bitmask=bitmask/2;
    }
}

int main() {
    unsigned char* cp;
    float f=43;
    cp=reinterpret_cast<unsigned char*>(&f);
    for (int byte=0;byte<4;byte++) {
        cout << "Byte " << byte << "=";
        printbinary(*(cp+byte));
        cout << endl;
    }
}

```

## Enumerations

[\[ back to contents \(#Contents\) \]](#)

Enumerations are another way to make code easier to read. Earlier we had an array of integers created using `int month[12]`; If we wanted to set the July value to 31 we could do

```
month[6]=31;
```

(remember, element indexing begins at zero). Alternatively we could use enumerations.

```
enum Month {Jan, Feb, Mar, Apr, May, Jun, Jul, Aug, Sep, Oct, Nov, Dec};  
month[Jul]=31;
```

`Month` is a new variable type whose only legal values are `Jan, Feb ... Dec`. These values are just aliases for 0,1 ... 11, but they're easier for humans to read.

## Writing to files [\[ back to contents \(#Contents\) \]](#)

To be able to use the file-writing facilities you need to add

```
#include <fstream>
```

to the top of your file. To illustrate writing to files, we'll write the values of the `month` array (one value per line) into a file called `myData`. Notice that the line that writes to the file is similar to the use of the `cout` command when writing on the screen

```
ofstream fileOut; // create a variable to store info about a file  
fileOut.open("myData"); // try to open the file for writing  
if (fileOut.good())  
{  
    for(int i=0; i<12; i++)  
    {  
        fileOut << month[i] << endl; //write to the file  
    }  
    fileOut.close();  
}  
else  
{  
    // error opening file  
}
```

The `good()` function returns `false` if there is an error.

## Semicolons [\[ back to contents \(#Contents\) \]](#)

By now (and with the help of the compiler's error messages) you've probably gained a lot of experience about where semi-colons are needed. Note that

```
if(i==3)
```

**doesn't** need a semi-colon after it because it's not a completed statement. Unfortunately, though it doesn't require a semi-colon, it's not illegal to have one. The following is legal C++

```
int i=0;
```

```
if(i==3); {
    cout << "i is 3" << endl;
}
```

and will print out `i is 3` even though `i` is 0. Why? Well, the `if(i==3)` code is an incomplete construction. It expects a statement after it, and in this context a semi-colon is a null-statement. The following layout shows more clearly what happens.

```
int i=0;
if(i==3) {
    ;
}
cout << "i is 3" << endl;
```

The `cout` line is run every time because it's not being controlled by the `if`. There's a similar risk with `while`. The compiler won't complain that the following code runs forever

```
int i=0;
while (i<10); {
    i++;
}
```

The trouble is that the `i++;` line isn't inside the body of the loop, so `i` is always 0. The following layout shows more clearly what happens

```
int i=0;
while (i<10) {
    ;
}
i++;
```

See the [I'm confused about commas and semi-colons](http://www-h.eng.cam.ac.uk/help/tpl/languages/C++/FAQ.html#1.1) (<http://www-h.eng.cam.ac.uk/help/tpl/languages/C++/FAQ.html#1.1>) frequently-asked-question for details.

## Exercise 8 - Measuring program speed [\[ back to contents](#)

(#Contents)

C++ has a function to sort items. We'll investigate how the time taken to sort items depends on `n`, the number of items. Is the time proportional to `n`? `n2`? The claim is that it's proportional to `n*logn(n)`, but we'll see.

The program below uses some commands you've not seen before, but they'll be useful to you next term and/or next year. Like next term, we're providing you with a program that works but is incomplete. The new features involve

- *Timing* - We've provided a routine you can use to time parts of your programs. It returns the number of microseconds that have elapsed since 1st Jan 1970. Don't worry about how it works.
- `vector` - C++'s `vector` is a more sophisticated version of an array. You'll be using it in the 2nd year. We're using it here because it makes the code simpler. You won't need to change any of the lines that use `vector` in this code. Just note that you read values from a `vector` just as you'd read them from an array.

Here's a program

```
#include <iostream>
#include <vector>
#include <sys/time.h>
#include <cmath> // needed for log
```

```

#include <iomanip>    // needed for setw
#include <algorithm> // needed for sort
using namespace std;

long microseconds();

// This function returns microseconds elapsed since the minutes value
// last changed - good enough for now
long microseconds() {
    struct timeval tv;
    gettimeofday(&tv,0);
    return 1000000*tv.tv_sec + tv.tv_usec;
}

int main() {
    long thelength=10;
    long thetime;

    // Create a vector called numbers big enough to store 'thelength' ints
    vector<int> numbers(thelength);

    // After the next lines, the vector number will contain
    // the integers 0 to thelength-1
    for (int i=0;i<thelength;i++) {
        numbers[i]=i;
    }

    // Randomize and output the values
    random_shuffle(numbers.begin(),numbers.end());
    for (int i=0;i<thelength;i++) {
        cout << numbers[i] << endl;
    }

    // Sort and output the values
    sort(numbers.begin(), numbers.end());
    for (int i=0;i<thelength;i++) {
        cout << numbers[i] << endl;
    }

    // Output the results, formatting so that each number is in
    // a column 10 characters wide.
    // First print the column headings
    cout << "    Length      Time    Time/n    Time/(n*ln(n))" << endl;
    cout << setfill (' '); // fill the gaps with spaces
    cout << setw (10) << thelength << endl;
    // The next line removes the contents of 'numbers' ready for
    // the next iteration of the loop that you're going to write
    numbers.clear();
}

```

Compile and run this code. You'll see a list of unsorted numbers, a list of sorted numbers, then part of a table.

First, comment out the lines that print the numbers (later you won't want to watch 10,000,000 numbers being printed out). Then complete the line of the table by timing the `sort` function (call `microseconds()` twice and find the difference between the 2 results). Time *only* the `sort` function - don't time the setting-up code too. It should take less than 10 ms. Then complete the final 2 columns of the table. If `Time/N` comes out to 0, you need to read the [More Types](http://www-h.eng.cam.ac.uk/help/tpl/languages/C++/1AComputing/Mich/#MoreTypes) (http://www-h.eng.cam.ac.uk/help/tpl/languages/C++/1AComputing/Mich/#MoreTypes) section again. Your output should be something like

Length	Time	Time/n	Time/(n*ln(n))
10	8	0.8	0.347436

Then using an appropriately placed `for` loop add a completed line of data to the table for lengths of 100, 1000 ... 10,000,000. You'll need to move the line that prints the column headings so that they only appear once.

Which length has the smallest time-per-element value? Is the time proportional to  $n \cdot \log_n(n)$ ? If  $\text{Time} = \text{constant} \cdot n \cdot \log_n(n)$ , what's the mean *constant* for the values of  $n$  you used? To find the mean,

- Create a variable to store the cumulative total of the values to want to find the mean of. Initialise it to 0.
- Each time you go round your loop, add a new value to the cumulative total.
- Once you've finished going round the `for` loop, work out the mean and print it on screen.

Finally, change the code so that the table is written into a file called "ex8table". When your program correctly produces this file, get the demonstrator to mark your last 4 programs. We'll be checking to see that you've produced the file, so don't delete it afterwards.

## More exercises [\[ back to contents \(#Contents\) \]](#)

You should try at least some of these before next term. The earlier ones are easier.

- Rewrite all the earlier exercises that used `while` loops so that they use `for` loops instead.
- Write a function that converts degrees to radians
- Write a function that prints the primes less than 100. Use `for` loops rather than `while` loops.
- Write a function that given an integer prints out its prime factors. Use `for` loops rather than `while` loops.
- Write a function that given 2 integers prints out their highest common factor
- Write a function that given 2 integers prints out their lowest common denominator
- By trial and error, find all the Pythagorean triples where the integers are less than 100 ( $a, b, c$  is a Pythagorean triple if  $a^2 + b^2 = c^2$ ). It's easy to eliminate duplicates (4 3 5 is a duplicate of 3 4 5). It's rather harder to eliminate multiples (e.g. 6 8 10 is a multiple of 3 4 5), but the whole program should be less than 20 lines long.
- Pick a number. If it's even, divide by 2. If it's odd multiply by 3 and add 1. Continue this until you reach 1. E.g. if you start with 3, the sequence will be 3-10-5-16-8-4-2-1. Write a program to find out which integer less than 100 produces the longest chain.
- Ask the user to type an integer in the range 1 to 7. If the user types something invalid, ask them to try again. Then print the day of the week corresponding to the number - print "Sunday" if the user types 1, etc.
- The zeta function,  $\zeta(s)$ , can be evaluated by summing terms of the form  $1/i^s$  for  $i$  running from 1 to infinity. It can also be evaluated by multiplying terms of the form  $(1/(1-1/p^s))$  where  $p$  runs through all the primes. See how true this is for series of 10 terms then 100 terms.
- Simulate a situation where you keep asking different people their birthdays until you find 2 people with the same birthday. Assume all years have 365 days. Make the code into a function so that you can run it many times and find an average for the number of people you need to ask.
- Big numbers are often printed with commas after every 3 digits - e.g. 1,345,197. Write a function which is given a `float` and prints out the number with commas. You'll need to find out how to convert numbers to strings - use the WWW!
- Create an array to represent an 8x8 chessboard. In chess a Bishop can move as far as it likes along diagonals. Write a program to count the number of places a Bishop can move to for a particular starting position. From which starting positions does it have the greatest choice of moves?
- In chess, a knight moves in an 'L' shape (2 squares along a row or column, then one square left or right). Write a program to count the number of places a Knight can move to for a particular starting position.
- Starting at any square on a chessboard, move the knight randomly, and keep moving until it lands on a square it's already visited. How many moves does it do on average?
- Write a function that adds 2 vulgar fractions and prints their sum as a vulgar fraction. E.g. its prototype could be

```
void addfractions(int numerator1, int denominator1, int numerator2, int denominator2);
```

so that calling it as `addfractions(3,4,5,6)` will make it output something like 38/24, or better still 19/12, or even  $1 + 7/12$ . You could create a class called Fraction

- Write a program that simulates the rolling of 10 dice 10000 times. Display a frequency table of outcomes (the outcome being the sum of the 10 dice) - e.g.

Outcome	Frequency
1	0
2	0
...	
60	1

(this was the final exercise in the Mich 2008 term)

- For the program below try to explain why the lines of output aren't all the same

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

int main() {
    cout << 4.0/3.0 -1.0/3.0 -1.0 << endl;
    cout << 4.0/3.0 -1.0 -1.0/3.0 << endl;
    cout << 4/3      -1    -1/3<< endl;
}
```

## Useful Links [\[ back to contents \(#Contents\) \]](#)

- [pre-lab briefing notes](http://www-h.eng.cam.ac.uk/help/tpl/work/1AC++/mich/slides/) (http://www-h.eng.cam.ac.uk/help/tpl/work/1AC++/mich/slides/)
- [CUED's C++ page](http://www-h.eng.cam.ac.uk/help/tpl/languages/C++.html) (http://www-h.eng.cam.ac.uk/help/tpl/languages/C++.html)
- [CUED's IA Computing Help](http://www-h.eng.cam.ac.uk/help/tpl/languages/C++/1AComputing.html) (http://www-h.eng.cam.ac.uk/help/tpl/languages/C++/1AComputing.html)
- [CUED Tutorial Guide to C++ Programming](http://www-h.eng.cam.ac.uk/help/languages/C++/c++_tutorial/) (http://www-h.eng.cam.ac.uk/help/languages/C++/c++\_tutorial/)
- [CUED's C++ Frequently Asked Questions](http://www-h.eng.cam.ac.uk/help/tpl/languages/C++/FAQ.html) (http://www-h.eng.cam.ac.uk/help/tpl/languages/C++/FAQ.html)
- [cplusplus.com's C++ Language Tutorial](http://www.cplusplus.com/doc/tutorial/) (http://www.cplusplus.com/doc/tutorial/)
- [Introduction to C++](http://ocw.mit.edu/OcwWeb/Electrical-Engineering-and-Computer-Science/6-096January--IAP--2009/CourseHome/index.htm) (http://ocw.mit.edu/OcwWeb/Electrical-Engineering-and-Computer-Science/6-096January--IAP--2009/CourseHome/index.htm) (MIT OpenCourseWare) - see especially the [Assignments](http://ocw.mit.edu/OcwWeb/Electrical-Engineering-and-Computer-Science/6-096January--IAP--2009/Assignments/index.htm) (http://ocw.mit.edu/OcwWeb/Electrical-Engineering-and-Computer-Science/6-096January--IAP--2009/Assignments/index.htm)

---

© 2011 University of Cambridge Department of Engineering

Information provided by [Tim Love \(tpl\)](#) and [Gabor Csanyi](#) (Last updated: July 2011)