MATH49111/69111 Scientific Computing

Dr. Paul Johnson

LECTURE CLASS: Thursday 9:00am - 10:00am
Alan Turing Ground Floor G-209

LECTURE/LAB CLASS: Friday 12:00pm - 2:00pm
Alan Turing Ground Floor G-105

OFFICE HOURS: Tuesday 10:30-11:30am

CONTACT DETAILS: Office - 2.107 Alan Turing Building
Direct tel - (0161 30)63660
E-mail - paul.johnson-2@manchester.ac.uk
Website - www.ma.man.ac.uk/~pjohnson/MATH49111

ASSESSMENT:

This course is entirely assessed by project work. There will be three mini-assignments, which when combined will count for 10% of your final mark for this module, to be handed in at the end of each of the first three weeks. There will be two large assignments, the first, counting for 40% is to be handed in at the end of week 8, and the second, counting for 50% is to be handed in at the end of week 12. Working code for each project must be demonstrated up to one week before the project is due to be handed in. DEADLINES MUST BE STRICTLY OBSERVED!!!
1 Introduction

Computers are an essential tool in the modern workplace. They are ideally suited to the processing, analysis and simulation of data. In order to use a computer to solve a particular problem, however, we must generate a list of instructions for the computer to follow, a task known as programming. For common tasks, the lists of instructions will almost certainly have been created by somebody else and we can use their work to solve our problem. For example, Microsoft Excel contains the instructions to perform a huge number of standard tasks. Nonetheless, Excel cannot do everything; and for unusual, or new, tasks a new set of instructions must be written. The aim of this course is to provide you with the necessary skills to generate your own lists of instructions so that you can use computers to perform (almost) any task.

A programming language is a set of keywords and syntax rules that are used to “tell” the computer what you want it to do. A list of instructions written in a programming language is called a program and is most often created using text editors or the text editor component of an integrated development environment. Ultimately, these instructions must be translated from the programming language into the native language of the computer (assembly instructions). The translation is performed by specialised programs known as compilers. In order to use a programming language you must have a compiler for that language installed on your computer.

At the time of writing, there are hundreds, if not thousands, of different programming languages, each with different strengths and weaknesses. The choice of programming language is driven in part by the nature of the project. There will usually be some compromises between efficiency, and how easy it is to write code. Matlab for instance, can quickly find the inverse of a matrix, but it will certainly not be the quickest way to get the answer.

C++ is perhaps best described as middle-level, computer-programming language. It allows for direct manipulation of bits, bytes and addresses, a feature of assembly languages, but does not sacrifice structure or portability. In contrast to high level languages, such as BASIC, FORTRAN and Pascal, C++ is not very fussy about converting between data types and performs relatively few (i. e. hardly any) run-time error checks. C++ demands greater responsibility from the programmer, but can provide more power and exibility than higher level languages.

In these notes, we shall discuss how to use C++ to allow the development of powerful and efficient solutions to complex problems. It should be noted that as a C++ programmer you will
be highly portable, not locked into a single operating system or package. As such we show to write C++ within both commercial and open source packages, on different operating systems, or simply within a text editor.

1.1 The programming work cycle

The compiler is responsible for checking that what you have written is legal; i.e. that you have obeyed all the syntax rules and not violated any restrictions in the language. When programming, a large amount of time is spent correcting your program in response to compiler errors.

Once compiled, or built, the program must then be run, or executed, at which point the computer will carry out the tasks specified in the program. Just because a program compiles, however, does not mean that it will run. It is perfectly possible to write a syntactically correct program that tries to perform an illegal action, for example dividing by zero. These run-time errors, as opposed to compile-time errors, are another source, or rather sink, of development time.

The basic work cycle when writing computer programs is illustrated below:

For complex projects, the work cycle can be simplified by using an integrated development environment (IDE), such as Microsoft Visual Studio, or the open source package Netbeans. The development environment contains a text editor, for writing programs; a compiler, for building the programs; and a whole range of debugging tools for tracking down problems during the compilation and/or execution of the program. For the uninitiated, however, IDEs can seem overwhelming with a huge range of options that are not required for simple projects.

1.2 The simplest C++ program

Every valid C++ program must “tell” the compiler where its set of instructions starts and finishes. Hence, every C++ program must contain a function, that contains every instruction to be performed by the program. The required function is called main() and the simplest C++ program is shown below:
int main() {}  

The program is very boring because the function contains no instructions, but it will compile and it will run. The keyword `int` indicates that the function `main()` will return an integer (a whole number) when it has completed all its instructions. In C++, sets of instructions are grouped together by braces, sometimes called curly brackets, `{}`; everything between the braces following `main()` will be executed while the program is running. The round brackets are used to specify arguments to functions. It is possible to write programs using only the main function, but to do so would fail to take advantage of the more powerful structural features of C++.

1.3 The C++ keywords

There are 32 keywords defined by the ANSI C standard, see Table 1. If you are programming in C, these are the only keywords that you have to remember and, in practice, the working vocabulary is about 20 words. The C++ standard defines an additional 32 keywords, shown in Table 2. C++ is a rapidly evolving language and the number of keywords may still change in the future. Case is important in C++ and keywords must be specified in lower case: e.g. `else` is a keyword, but `Else`, `ELSE` and `ELSE` are not.

<table>
<thead>
<tr>
<th>auto</th>
<th>double</th>
<th>int</th>
<th>struct</th>
</tr>
</thead>
<tbody>
<tr>
<td>break</td>
<td>else</td>
<td>long</td>
<td>switch</td>
</tr>
<tr>
<td>case</td>
<td>enum</td>
<td>register</td>
<td>typedef</td>
</tr>
<tr>
<td>char</td>
<td>extern</td>
<td>return</td>
<td>union</td>
</tr>
<tr>
<td>const</td>
<td>float</td>
<td>short</td>
<td>unsigned</td>
</tr>
<tr>
<td>continue</td>
<td>for</td>
<td>signed</td>
<td>void</td>
</tr>
<tr>
<td>default</td>
<td>goto</td>
<td>sizeof</td>
<td>volatile</td>
</tr>
<tr>
<td>do</td>
<td>if</td>
<td>static</td>
<td>while</td>
</tr>
</tbody>
</table>

Table 1: The 32 C keywords

1.4 Syntax of the source code

The key elements of C++ syntax are shown below:
1 INTRODUCTION

<table>
<thead>
<tr>
<th>asm</th>
<th>export</th>
<th>overload</th>
<th>throw</th>
</tr>
</thead>
<tbody>
<tr>
<td>bool</td>
<td>false</td>
<td>private</td>
<td>true</td>
</tr>
<tr>
<td>catch</td>
<td>friend</td>
<td>protected</td>
<td>try</td>
</tr>
<tr>
<td>class</td>
<td>inline</td>
<td>public</td>
<td>typeid</td>
</tr>
<tr>
<td>const_cast</td>
<td>mutable</td>
<td>reinterpret_cast</td>
<td>typename</td>
</tr>
<tr>
<td>delete</td>
<td>namespace</td>
<td>static_cast</td>
<td>using</td>
</tr>
<tr>
<td>dynamic_cast</td>
<td>new</td>
<td>template</td>
<td>virtual</td>
</tr>
<tr>
<td>explicit</td>
<td>operator</td>
<td>this</td>
<td>wchar_t</td>
</tr>
</tbody>
</table>

Table 2: The 32 additional C++ keywords

- A semicolon is used to mark end of an instruction.
- Case is important, i.e. sin is not the same as SIN.
- Totally free form, lines and names can be as long as you like!
- Comments take the form /* C style comment */ or // C++ style comment.
- Groups of instructions are surrounded by braces {}.

The most important point to remember is that all statements must end with a semicolon ;. Often, forgetting a semicolon can cause a huge number of compilation errors, particularly if the omission is near the start of a program.

1.5 Practicalities: compiling and running C++ on Windows

Before it can be compiled, a C++ program must be saved as a file and, by convention, C++ programs are labelled by the filename extensions .cc, .cpp or .C. Large projects may, and probably should, be split into separate files, which can be compiled separately. The division of large projects into separate “compilation units” speeds up the re-compilation of the whole project if only one small part of the project has been changed. Keeping track of separate files and their interdependence is an area in which IDEs are extremely useful. The alternative is to use command-line compilation and keep track of all the different files yourself. On unix operating systems we can also use Makefiles. There will be an (optional) introduction to these in reading week.
A command line is, as the name suggests, a place where you can issue (type) commands. Rather than clicking an icon to start a program, you must type the name of the program and then press return. Before windowing environments, all computation was performed using the command line and it is still easier, and quicker, to use the command line for compiling very simple programs.

We consider both “simple” command-line compilation and compilation within an IDE (Visual Studio) for a simple program that merely prints the word Hello on the screen. The C++ instruction that performs this task is

```cpp
std::cout << "Hello" << std::endl;
```

### 1.5.1 Simple command-line compilation

The easiest way to write a small C++ program is to use a text editor, say Notepad, to generate a .cpp file that contains the required instructions and then to compile the program in a Command Prompt Window. The command for the Visual Studio C++ compiler is `cl /EHsc file.cpp`, where `file.cpp` is the file that contains your C++ program. The process is illustrated in Figure 1.

The result of the compilation process is an executable file hello.exe. In order to run the program one can either double click on the hello.exe program from the File Manager, or type `hello.exe` (and then return) into the command prompt, see Figure 1.

This method of compilation will work for all simple projects. The method can be made to work for more complex projects, but it becomes more and more difficult to do so. A better way of managing large projects is to use an IDE. Initially, this will seem more complicated than the simple method outlined above, but it does not become significantly more complex as the projects become larger. We shall outline how to use Visual Studio on the university clusters. Bear in mind that in order to enable full optimisation in Visual Studio you will need to purchase a licence. You may also choose to use netbeans on university clusters or at home, but the debugging will not work on university machines.
1.5.2 Visual Studio Compilation

Once Visual Studio has been started, it is used to write, compile and run the program. One major advantage of Visual Studio is that it automatically includes a number of libraries that are used to help your program interact with the Windows operating system. A library is a collection of functions that have already been written and compiled. Precisely which libraries should be included depends on the type of program that you are writing. If the program will use graphics or interact with the mouse it requires more libraries than a simple command line application. You can specify which libraries are included by choosing your project type when creating a new project. During this course, we shall consider only (Win 32) Console Applications.

The first time that you run Visual Studio you may be asked to select a default environment,
in which case you should select “Visual C++ Development Settings”. In Visual Studio 2003 (and earlier), the Start Page of Visual Studio contains a New Project button, which when pressed brings up the New Project dialog box, see Figure 2. In Visual Studio 2005, the same dialog box is accessed by clicking the Project: Create link in the Recent Projects window on the Start Page.

For simple C++ programs, the project should be a Visual C++ Console Application, also called a Win32 Console Application, located in the Win32 submenu of the Visual C++ project templates. Filling in a name, e.g. **Hello**, for the project in the dialog box and double-clicking on the Console Application icon from the Templates window (or clicking Open in the dialog box after selecting the Console Application icon) will bring up the Application Wizard. For the default settings, simply press Finish and Visual Studio will then create a number of files, which should appear in the Solution Explorer window on the side of the screen. For information you can double-click on the **ReadMe.txt** file. Double-clicking on **Hello.cpp** (the actual C++ program)
brings up a short section of C++ code, see Figure 3. In order to write our own program, we comment out the pre-created instruction in the function `int _tmain()` and add our instruction. The program may then be compiled by going to the Build menu and choosing the Compile option. Alternatively, we can choose to run our program by selecting Start from the Debug menu. If the program has been modified without compiling it a dialog box pops up, see Figure 4, pressing yes causes the program to be compiled and then run. Thus, the entire write, compile and run process takes place within Visual Studio.

When using Visual Studio on university clusters there are problems with licencing which means that you cannot compile your programs to run on the P drive. There is a way around this (although very tedious) which I have put up on my website:
just like their windows counterparts, a C++ program must be saved as a file and, by convention, 
C++ programs are labelled by the filename extensions .cc, .cpp or .C. This is a little easier to 
do on unix as the file extensions are not automatically added. Large projects may, and probably 
should, be split into separate files, which can be compiled separately. The division of large 
projects into separate “compilation units” speeds up the re-compilation of the whole project if 
only one small part of the project has been changed. Keeping track of separate files and their
interdependence is an area in which IDEs are extremely useful. The alternative is to use command-line compilation and keep track of all the different files yourself. On unix operating systems we can also use Makefiles. There will be an (optional) introduction to these in reading week.

It can often be quicker to compile and run programs on unix from the command-line in a terminal. Here we shall consider compilation from the terminal and compilation within an IDE (Netbeans) for a simple program that merely prints the word Hello on the screen. The C++ instruction that performs this task is

\[ \text{std::cout} \text{ "Hello" } \text{std::endl;} \]

1.6.1 Simple command-line compilation

The easiest way to write a small C++ program is to use a text editor, say emacs, to generate a .cpp file that contains the required instructions and then to compile the program in a terminal. The command for the gnu C++ compiler is simply `c++ file.cpp`, where `file.cpp` is the file that contains your C++ program. The process is illustrated in Figure 5.

![Figure 5](image)

Figure 5: An illustration of simple command-line compilation. An emacs window containing a simple C++ program, `hello.cpp`, that will print the word Hello on the screen; and a terminal showing the compilation of the program `hello.cpp` and its execution.

The result of the compilation process is an executable file `a.out`. In order to run the program simply type `./a.out` (and then return) into the terminal, see Figure 5.
This method of compilation will work for all simple projects. The method can be made to work for more complex projects, but it becomes more and more difficult to do so. A better way of managing large projects is to use an IDE. Initially, this will seem more complicated than the simple method outlined above, but it does not become significantly more complex as the projects become larger. We shall outline how to use Netbeans on the university clusters.

1.6.2 Netbeans Compilation

Like Visual Studio, netbeans is used to write, compile and run the program. One major advantage of an IDE such as netbeans is that it can be used to help integrate libraries into your code with minimum effort. A library is a collection of functions that have already been written and compiled. Precisely which libraries should be included depends on the type of program that you are writing. If the program will use graphics or interact with the mouse it requires more libraries than a simple command line application. You can specify which libraries are included by choosing your project type when creating a new project. Netbeans has the capability to work with the QT libraries which provide cross platform access to graphics etc. During this course, we shall consider only simple C++ Applications.

For simple C++ programs, the project should be a C/C++ Application, located in C++ submenu of the netbeans project templates. Fill in a name, e.g. Hello, for the project in the dialog box and click finish. This will create an empty project, into which we must add our C++ files. Right click on source files and choose new C++ main file. Simply call the file main and click finish. A empty main file should now appear, into which we can enter our command, see figure 6.

The program may then be compiled by clicking the Build icon (hammer). We can then run the program by clicking the run icon, a green triangle. Thus, the entire write, compile and run process takes place within netbeans. There is also a guide on my website on how to create the hello world program:

\[\text{http://www.maths.manchester.ac.uk/~pjohnson/HowTo/netbeans-on-unix}\]

A complete description of the features of netbeans is well beyond the scope of this introductory course. A vast amount of information is provided in the on-line help documentation.
1.7 Debugging with your chosen IDE

I have written some hints on how to debug your programs on my website in the HowTo section:

http://www.maths.manchester.ac.uk/~pjohnson/MATH49111

Figure 6: An illustration of simple NetBeans compilation. The terminal showing the output from the program is shown on the right, while in the background we can see the program main.cpp.
2 Getting started with C++

2.1 Data types

Almost all computational tasks require the manipulation of data, e.g. finding the maximum of a set of numbers, printing an e-mail, modelling the growth of the stock market, etc. In C++, the particular type of each datum must be declared before it can be used. It is possible to create your own “custom” data types, but for simple tasks the seven built-in data types (`char`, `wchar_t`, `bool`, `int`, `float`, `double` and `void`) are sufficient. Table 3 shows the type of data represented by the first six keywords.

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>char</code></td>
<td>a single character (letter)</td>
</tr>
<tr>
<td><code>wchar_t</code></td>
<td>a wide character (letter)</td>
</tr>
<tr>
<td></td>
<td>(used for characters from non-english alphabets)</td>
</tr>
<tr>
<td><code>bool</code></td>
<td>a boolean datum (true or false)</td>
</tr>
<tr>
<td><code>int</code></td>
<td>an integer datum (whole number)</td>
</tr>
<tr>
<td><code>float</code></td>
<td>a floating-point datum</td>
</tr>
<tr>
<td></td>
<td>(a real number with six digits of precision)</td>
</tr>
<tr>
<td><code>double</code></td>
<td>a double-precision, floating-point datum</td>
</tr>
<tr>
<td></td>
<td>(a real number with ten digits of precision)</td>
</tr>
</tbody>
</table>

Table 3: The (non-void) C++ built-in data types.

The `void` data type is rather special and is used to declare a function that does not return a value (a subroutine).

2.2 Variables

A variable is a named location in memory, used to store something that may be modified by the program. All variables must be declared before they are used. You can call variables anything you like, provided that the name you choose does not start with a number, is not a C++ keyword and is not the same as any other variable within the same scope. It is usually a good idea to use descriptive names that indicate what the variable represents. Example declarations are shown below:
Note that commas may be used to separate variables of the same type and that each line ends with a semicolon. The initial value of a variable may be assigned when the variable is declared. Uninitialised variables can be a source of mysterious errors, so it’s a good idea to initialise your variables if practical:

```cpp
int i=0,j=1,k=2;
double a=12.35648;
char ch='a';
```

Note that characters must be enclosed in single quotes ‘’.

### 2.2.1 The scope of variables

C++ permits the declaration of variables anywhere within your program, but a variable can only be used between the braces {} within which it is declared. The region in which the variable can be used is known as its **scope**. An advantage of local scopes is that you can use the same name for variables within different regions of the same function. The following two example codes illustrate these ideas.

```cpp
int main() {
//Declare a in new scope
{int a = 10;}
//Declare a in current scope
int a=10;

//Re-declare a in scope (illegal)
//a used outside scope (illegal)
//a used in scope (ok)
a = 20;
}
```

Neither of the two codes will compile. The code on the left fails on the last line of `main` with the error `'a' : undeclared identifier` because `a` has not been declared in scope. The code on the
right fails on the fourth line of `main` with the error `a : redefinition; multiple initialization` because the variable `a` cannot be declared twice in the same scope.

### 2.2.2 Arrays

An array is a just like an array in maths: a collection of variables of the same type, called by the same name, but each with a different index. The standard way to define an array is to enclose the dimension in square brackets `[ ]` after the variable name.

```cpp
int x[100]; // 100 integer array
double y[300]; // Array of 300 doubles
```

Arrays are initialised by using braces to define the array and commas to separate the individual array entries:

```cpp
double x[3] = {1.0, 2.5, 3.7};
```

The individual entries of an array are accessed by adding a single number in square brackets after the variable name. The array `double x[3]` has the entries `x[0] (= 1.0)`, `x[1] (= 2.5)` and `x[2] (= 3.7)`. **Important note:** In C++, array indices start from 0.

Multidimensional arrays are declared and initialised in an obvious way:

```cpp
double matrix[10][10];
int 3dtensor[3][3][3] = {{{1,2,3},{4,5,6},{7,8,9}}};
```

The maximum number of dimensions, if any, is determined by the compiler. Note that arrays defined in this way are allocated in the stack, an area of memory designed to store variables with short lifetimes. The size of the stack is set by the operating system. An area of memory that is designed to store longer-lived variables is known as the heap, but can only be accessed via dynamic allocation.
2.3 Manipulating Data — Operators

We can now create and initialise variables, but how do we modify the data? The answer is to use operators, which, as the name suggests, operate on the data. The built-in operators in C++ may be broadly subdivided into four classes arithmetic, relational, logical and bitwise.

You have already been introduced to the assignment operator, =. The most general form of assignment is

\[ \text{variable} = \text{expression}; \]

Note that C++ will convert between data types in assignments. The code shown below will compile without complaint.

```cpp
int i=1;
char ch;
double d;

ch = i; // Conversion from int to char
d = ch; // Conversion from char to double
```

The compiler will usually do the right thing, for example converting from double to integer should give the integer part of the result. That said, it is unsafe to rely on automatic conversion, and doing so can give rise to funny errors. Be sure to check data types in expressions carefully.

2.3.1 Arithmetic Operators

Table 4 lists the C arithmetic operators, which are pretty obvious apart from the last three, %, -- and ++. The modulus operator, %, gives the remainder of integer division, it cannot be used on floating point data types. The increment and decrement operators may seem strange at first: ++ adds 1 to its operand and -- subtracts 1. In other words, \( x = x + 1; \) is the same as ++x; and \( x = x - 1; \) is the same as x--; Increment and decrement operators can precede or follow the operand and there is only a difference between the two when used in expressions.

\[ x = 10; \]
\[ y = ++x; // Increments x and then assigns y; i.e. x = y = 11 \]
Table 4: The C++ arithmetic operators

- Subtraction
+ Addition
* Multiplication
/ Division
% Modulus
-- Decrement
++ Increment

x = 10;
y = x++; // Assigns y and then increments x; i.e. x = 11, y = 10

Arithmetic operators are applied in the following order:

First

++ --

- (unary minus)

* / %

Last

+ -

Parentheses () may be used to alter the order of evaluation by forcing earlier evaluation of enclosed elements. i.e. 2 + 3*5 = 17, but (2+3)*5 = 25.

2.3.2 Relational and logical operators

Relational and logical operators rely on concepts of false and true, which are represented by integers in C++. False is zero and true is any value other than zero. Every expression involving relational and logical operators returns 0 for false and 1 for true. Table 5 shows all the relational and logical operators used in C++.

Examples of relational operations and their values are

10 > 5 // True (1)
19 < 5 // False (0)
2.3.3 Shorthand operators

C++ uses a convenient (well some would, and do, say obscure) shorthand to replace statements of the form $x = x + 1$,

$\begin{align*}
&x += 10; /* is the same as */ x = x + 10; \\
&x -= b; /* is the same as */ x = x - b;
\end{align*}$

This shorthand works for all operators that require two operands and is often used in professional C++ programs, so it is well worth taking the time to become familiar with it. In fact, shorthand operators are slightly more efficient because they avoid the need to create temporary variables.
2.4 Talking to the world — Input and Output

No matter how wonderful your program is in isolation, at some point it’ll need to interact with the outside world. I/O or input and output is not part of the standard C++ keywords. The appropriate functions are present in libraries that can be incorporated into any C++ program. The necessary library is called iostream and is included using the compiler directive \#include<iostream>. In fact, Visual Studio automatically includes the appropriate libraries for you when it creates the Console Application template.

An example of simple I/O is shown below

```cpp
#include <iostream>

int main()
{

    int i; //Declare an integer variable

    std::cout << "This is output.\n"; //print a string

    std::cout << "Enter a number: ";
    std::cin >> i; //read in a number

    // output the number and its square
    std::cout << i << " squared is " << i*i << "\n";
}
```

std::cout is a stream that corresponds to the screen and std::cin is a stream that corresponds to the keyboard. The \<< and \>> operators are output and input operators and are used to send output to the screen and take input from the keyboard. Several output operators can be strung together in the same command. The special character '\n' represents a newline. This program reads into a variable, i, that has been declared to be an integer and what you type at the keyboard is automatically converted into an integer. If the input data is not of the correct type very strange things can happen, see below.
21

% ./a.out
This is output
Enter a number 5
5 squared is 25

% ./a.out
This is output
Enter a number 1.45
1 squared is 1

% ./a.out
This is output
Enter a number a
1 squared is 1

### 2.4.1 A note on namespaces

You may be wondering about the meaning of the prefix `std::`. The answer is that `std` is a C++ namespace. When writing large projects it is almost impossible to think of unique names and it’s easy to accidentally call two distinct things by the same name. In an attempt to prevent this, C++ allows the grouping of functions and data into named collections — namespaces. All functions and classes defined in the standard libraries are in the standard namespace, `std`. In order to tell the compiler that we want to use `cout` from the namespace `std` the syntax is `std::cout`. An alternative is to place the statement `using namespace std;` at the top of our program, see later examples, in which case the standard namespace will be searched for any unknown names and we do not need to use the `std::` prefix at all.

### 2.4.2 File I/O

File I/O is very similar to the I/O system described above and is also based upon streams. An extra header file must be included for file I/O, `#include <fstream>`. An input stream uses the `ifstream` class and an output stream uses the `ofstream` class. The following program writes a
simple text file that lists the integers 1 to 10 in one column and their squares in the next column.

```cpp
#include <iostream>
#include <fstream>

int main()
{
    using namespace std;
    ofstream out("test.out"); // Open an output file called test.out
    // The output stream is called out
    //Error check:
    //If the file has not been opened then out will be zero
    if(out==0)
    {
        cout << "Cannot open file test.out\n";
        throw; // Throw an exception (end program)
    }

    //A loop in which i takes the integer values 1 to 10
    for(int i=1;i<=10;i++)
    {
        //Print the value of i and i squared to the output file
        out << i << " " << i*i << endl;
    }

    //Close the output file
    out.close();
}
```

We test whether the file has actually be opened by using an if statement, see §3. If the file has not been opened, the error is reported by using the keyword throw, which immediately exits the main function and returns control to the operating system. The object endl is an output stream
modifier that outputs a newline and flushes the stream, i.e. writes everything to disk.

In fact, opening a file does not have to be done when the stream is declared. The command
\begin{verbatim}
out.open("test.out");
\end{verbatim}
can be used anywhere in the body of the function.