CodeLab I - CCO4000-20
Programming Fundamentals
C++ Workbook

Creative Computing
The School of Creative Industries
Bath Spa University

Contents

Getting Started

Installing an IDE

Class demonstrations are taught using Visual Studio, which is installed on all the G.32 PCs. If using your own laptops in class you are welcome to use any IDE of your choice. For example if you are on Mac we'd recommend using XCode. To practice programming outside of class you should ensure you have an IDE installed on your home computer. If you are unsure of how to install an IDE contact your tutor for help.

If you are struggling to install a compiler on your own computer online ones are also available which allow you to run and save your code (so there's no excuse for not coding outside of class).

You can find information on how to start a project in Visual Studio or Xcode in the guides provided on Aura / your GitHub repository.

Using this workbook

Use this workbook in combination with the session slides and notes for full information on everything covered through CodeLab. The chapters and content are ordered in line with the classroom content allowing you to follow along before putting what you've learned into practice with the workbook exercises. You should aim to complete these exercises in order as without a firm understanding of previously introduced techniques you may struggle to complete the latter more advanced exercises.

Practicing outside of class

Programming is a difficult skill to master and you will struggle to get to grips with it by coding in the classroom sessions alone. You need to practice coding between classes to ensure you do not get left behind. This workbook should be used to practice both inside and outside of class, allocating yourself time in the week to complete exercises not completed during the class. 2 - 4 hours (or more) of additional practice each week will make a world of difference to your code. A solutions booklet to all exercises is available on Aura / your GitHub repository so you can check your answers, or need help if you get stuck. However, do persevere and try to solve the challenges yourself before resorting to the solutions.

Further Reading & Bonus Exercises

If you have previous programming experience you will find advanced notes with further reading resources, extension problems to the exercises and more advanced bonus exercises in this workbook to push your knowledge. You will also find an additional bonus exercises workbook on Aura / your GitHub repository, which provide longer programming tasks that can be solved in a variety of ways. Tackle them in any order you wish. Each will require a mixture of the techniques introduced through the workshops so far (selection, loops, arrays, functions, etc). Some of these exercises will take a number of hours / days to complete and are there to help push the boundaries of your C++ knowledge.

Using GitHub

Throughout CodeLab we will be using GitHub to keep track of the code we create. You have been provided with a GitHub repository that includes all the learning materials for CodeLab and is the place where you should save your solutions to the code exercises in this workbook. Your tutor will regularly review the code you post on GitHub in order to provide feedback on your work to help you improve. The more you post to GitHub the more feedback that can be offered.

GitHub will also be used for the submission of assignment code. Guides on using GitHub can be found on Aura and if you have any questions let your tutor know.

1-2-1 Tutorials

If you are struggling with any of the concepts introduced in class and would like some additional programming help 1-2-1 tutorials are always available. To book a tutorial visit:

https://jhobbs.youcanbook.me

Spot an error?

If you spot an error in this workbook please let me know via j.hobbs@bathspa.ac.uk

Chapter 1 - Introduction to C++

Coding Conventions

The importance of Coding Conventions is crucial. There is nothing worse in programming then opening up a C++ file (or HTML5 or CSS etc.) and being confronted with the code equivalent of an overturned filing cabinet. Keeping your code neat and tidy not only helps you find your way around, but helps others understand your work when looking for help. Neat code can save you hours when trying to fix errors and its importance should not be underestimated (which is why its part of the marking criteria for the Skills Portfolio and Utility App assessments!). Here are six tips that should help you keep your code organised so you can make sense of it when you return to it at a later date.

#1 - Programme Descriptors

Add a couple of commented lines right at the very top of your programme that describes, in the simplest terms, what the program does.

#2 - Comments

Beginners should annotate their code with comments to remind them what certain commands do. For example, the first time you type

```
cout << "Hello World" << endl;
```

it wouldn't hurt to place a comment next it that says…

```
// This is how you print to the console in C++.
```

As your programmes become more complex, you can use comments to make it easier to find certain code blocks, or remind you to finish certain coding tasks. Comments act like bookmarks in this sense.

#3 - Variable Names

Variable names should be unambiguous and where possible, short. For example if I was writing a programme that asks for a user's name, I would save their input in a variable called usersName. Seems obvious, but you'd be surprised how often variables names such as 'a' or 'x' are used. This is lazy and makes understanding what is happening in the program much harder

#4 - CamelCase

If you use more than one word for a variable name (e.g. myInteger), make sure you capitalise each word. Most people only capitalise from the second word onwards (e.g. myFavouriteFood).

#5 - Whitespace

Write code as if you were writing an instruction manual. Keep everything in line (unless you have to indent - more on this later), and only break up code blocks with blank lines if it makes sense to do so. Think of code blocks as paragraphs with short sentences that should be kept together until the next sentence is clearly the beginning of a new set of instructions.

#6 - Indentation

Indentation helps identify where code blocks begin and end. Code blocks in C++ are wrapped in a pair of curly braces { }, and the code inside the block should be indented by 1 tabbed space. If nesting a code block inside another this would be tabbed again. See the example below and don't worry at this stage if code doesn't make sense… it will soon!

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

int main() { // this is the start of the main program code block

    //code inside the block is indented by one tabbed space
    cout << "Hello, World!" << endl;
    cout << "This is a simple C++ program" << endl;

    if(6 < 7){//this is the start of a second code block
        //code inside this block is indented a further tabbed space
        cout << "6 is less than 7" << endl;
    }

    return 0;

} //this is the end of the main program code block
```

Variables & Data Types

A data type is a particular kind of data object, for example, a whole number (int), single character (char) or a decimal number (double). A variable's data type determines the values it may contain, plus the operations that may be performed on it.

A variable in computer programming is a value that can change. Variables are like little boxes that contain values. We put a value (e.g. 8) in a box and give it a tag, for example 'myInteger'. That way, whenever you want to access the number in the box, you just refer to the variable, 'myInteger'. The important thing is however, that you can change the value in the box, yet the name of the box still remains the same.

The core data types: int (whole number), char (single character), double (decimal number) are accompanied by similar types which expand or decrease their size and range. The choice between using similar data types is usually a matter of…

- Size of number expected. You may need to calculate large numbers
- Accuracy. If you need a certain level of precision when calculating numbers
- Managing memory resources. Memory can be saved if you only use the appropriate data type
- Size for the job. For example, declaring a variable for an individual's age can be achieved with short.

Within CodeLab we will typically just be using the basic core data types (int, char, double), but for reference the below table lists the data types supported in C++, including their size (amount of space they'll take up in memory) and their range.

Type	Typical Bit Width	Typical Range
char	1byte	-127 to 127 or 0 to 255
unsigned char	1byte	0 to 255
signed char	1byte	-127 to 127
int	4bytes	-2147483648 to 2147483647
unsigned int	4bytes	0 to 4294967295
signed int	4bytes	-2147483648 to 2147483647
short int	2bytes	-32768 to 32767
unsigned short int	Range	0 to 65,535
signed short int	Range	-32768 to 32767
long int	4bytes	-2,147,483,648 to 2,147,483,647
signed long int	4bytes	same as long int
unsigned long int	4bytes	0 to 4,294,967,295
float	4bytes	+/- 3.4e +/- 38 (~7 digits)
double	8bytes	+/- 1.7e +/- 308 (~15 digits)
long double	8bytes	+/- 1.7e +/- 308 (~15 digits)
wchar_t	2 or 4 bytes	1 wide character

Advanced Note

In C++11 the auto keyword was included to allow a variable data type to be inferred from the value its initialised with.

```
int myFirstInt   5; //explicitly declared integer variable
auto mySecondInt   6; //variable data type automatically inferred from the value assigned
```

For the declaration of simple variables with one of the core data types (int, char, double), it is best to be explicit and you would not use auto. Auto is typically used when your programs become much more complex and you are using complex types available in C++.

Further Information

https://en.cppreference.com/w/cpp/language/auto
https://www.learncpp.com/cpp-tutorial/4-8-the-auto-keyword/

Non-Primitive Data Types (Classes/Objects)

In C++ we also have access to non primitive data types. These are different to the above mention primitive types as they are objects obtained from core library classes (or custom made ones). These objects usually contain enhanced functionality beyond primitive data types that allow the objects to be manipulated.

Strings are one of the most widely used objects in programming. They hold a sequence of characters surrounded by double quotation marks “ on both sides and can contain any characters.

Example Strings: *“Hello”, “Hello World”, “Bye”*

Other non-primitive types we will use in future sessions are Arrays and Vectors, which act as containers for multiple variables/objects.

Input / Output Stream

The `#include <iostream>` directive written at the top of a C++ program provides us access to the C++ standard library input and output stream. We can use this functionality to get user input from the console and output information to the console.

We use `cin` and the extraction operator `>>` to get user input from the console

We use `cout` and the insertion operator `<<` to output information to the console.

Hello World

In programming Hello World is the traditional first program to write when getting started with any language. Below is the sample code for a basic Hello World program written in C++. Refer to the comments and code breakdown for an explanation of the main elements of the program, many of which you will require for every C++ program you write

```
#include <iostream> // preprocessor directive - include iostream file
using namespace std; // declare use of the standard namespace

int main(){ // define main function
	cout << "Hello World" << endl; // output statement

	return 0; //return statement
}
```

Code Breakdown

```
#include <iostream> // preprocessor directive - include iostream file
```

Preprocessor directive - anything beginning with a #hash is a preprocessor directive that instructs the compiler to process this information before compiling of the rest of the code. The #include instruction acts like a copy and paste command and tells the compiler to copy and paste the contents of the iostream header file to the top of our program. The iostream header file contains the declarations for the standard input-output library in c++. We need this to gain access to things like cout which is the standard library output command for outputting content to the console.

```
using namespace std; // declare use of the standard namespace
```

Using Namespace Std - The contents of iostream that we've just added are part of the standard C library which includes lots of useful predeclared functionality for c++ programs. All the elements of this library are declared within a namespace in this case the `std namespace`. To access its functionality we need to let the compiler know we are using this standard namespace. With the namespace declared if the compiler encounters any undeclared identifiers in the code (e.g. cout), it will check to see if they are present within the namespace. If it is the program will proceed, otherwise it will throw an error. Declaring the use of the standard namespace saves us having to explicitly declare its use every time we use functionality from the library. For example if without declaring use of the std namespace our cout statement in our Hello World program would need to be written like the example below with std explicitly declared before both `cout` and `endl` as the functionality for come from the standard library:

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

Advanced Note

When starting out with small programs declaring `using namespace std` at the beginning of our programs saves times as we can avoid the constant explicit declaration as described above. However, when you become more proficient and begin writing more advanced programs it is advised that you avoid its declaration at the start of the program and do explicitly declare the namespace before operations like `cout` each time. This helps avoid potential conflicts between different libraries you might be using that are using the same names for different operations.

```
int main(){ // define the main function
      cout << "Hello World" << endl; // output statement
       return 0; //return statement
}
```

The main function - All C++ programs start with the execution of a main function. It is essential to have a main function otherwise your program simply will not work, this is the first thing the compiler looks for when executing C++ code. The word main is followed by a pair of parentheses followed by the body of the main function which will be enclosed by curly braces. The function body includes all the code that we want our program to run. C++ programs are executed line by line in order. Each line is a statement which is an expression that can produce a result. Statements are always terminated by a semicolon. Omitting the semicolon is probably the most common error when writing code. Try removing the semicolon at the end of the hello world statement, a red line should appear on that line in the IDE to indicate an error. These red lines are useful for helping find errors when writing your code.

The first line in our main function is an output statement. `cout` is the standard output stream in c++. Our cout statement is saying insert the sequence of characters for “hello world” into the output stream. You will notice the two less than signs `<<` this is called the insertion operator. So we are inserting Hello World into the output stream.

We then insert something called `endl`. This is an instruction to insert a newline then flush the contents of the output stream.

Advanced Note

A new line can also be created by inserting the newline character `\n` within a string. For example:

```
cout << "Hello World \n";
```

The difference between `endl` and `\n` is `endl` flushes the output stream after adding the newline. Flushing the stream ensures the contents are outputted to the console in a timely manner. If you have several cout statements one after the other, it may not be necessary to flush the stream at the end of each `cout` statement. Therefore you could use the newline character instead and then on the final line use `endl` to ensure the stream is flushed.

```
cout << "Hello, World! This is the first line \n";
cout << "Here is a second line \n";
cout << "Here is a third line \n";
cout << "This fourth line will flush afterwards" << endl;
```

Further Information
https://www.geeksforgeeks.org/endl-vs-n-in-cpp/

The final line of the main function is the return statement. The return statement causes the main function to finish. Usually this is followed by a return code in our case 0, which is generally interpreted as the program worked as expected without errors.

Visual Studio Note

When running the program on Visual Studio you may think it is not working. This is because the console window exits as soon as the return statement is reached, which happens so quickly it appears the program does not run. To force the console window to stay open include the following line before the return statement. This line of code waits for input e.g. pressing enter.

```
cin.get();
```

Operators

An operator is in computer programming is a symbol that tells the compiler to perform specific mathematical, relational or logical operations.

This might seem complex, but in fact we use these symbols all the time. The most simple operators to understand are probably the mathematical or arithmetic operators.

Mathematical Operators

These include the following operators:

+	(additional)
-	(subtraction)
/	(division)
*	(multiplication)
%	(modulus or 'find remainder')

Here are some examples of mathematical operators in action. Note the use of variables here.

```
int a   5;
int b   7;
int c   a + b;
```

The value of the variable c is 12.

```
int d   20;
int e   10;
int f   d   e;
```

The value of the variable f is 10.

```
int numberOne   10;
int numberTwo   3;
int remainder   numberOne % numberTwo;
```

The value of the variable remainder is 1.

Concatenation

The + operator can also be used to concatenate or 'link' Strings. This means you can join one String to another using the + operator.

```
string h   "hello";
string i   " ";
string j   "world";
cout << h + i + j << endl;
```

This concatenates the word "hello" with a space " " and the word "world". The String printed to the console here would look this this: "hello world".

The Assignment Operator

The assignment operator should also be familiar to you.

It looks like this:

=

This simply means you are making one thing equal to something else. Here are some examples.

```
double a   10.5;
```

Here you are assigning the variable a the value 10.5 - note the use of *double* as the data type as we are dealing with a decimal number.

```
bool myBool   false;
```

Here you are assigning the variable myBool the value false.

Exercises

DataTypes

Type out the code below and add the correct data types to complete the variables

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

int main() {
    firstNumber   1;
    iCanCode   true;
    hopefulGrade   'a';
    myDecimal   1.0;
    minimalSentance   "y";
    keyMash   13213123;
    mysteryDataType   5.6f; //Use Google to work this out

    cin.get(); //keeps console window open in Visual Studio
    return 0;
}
```

Note: Copy and paste from the workbook will cause errors in the IDE

Division Fix

This programme doesn't quite work as expected. Can you provide a simple fix?

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

int main() {
    int numberOne   50;
    int numberTwo   7;
    cout << numberOne/numberTwo << endl;

    cin.get(); //keeps console window open in Visual Studio
    return 0;
}
```

Note: Copy and paste from the workbook will cause errors in the IDE

Untidy Code

Tidy up the code to make it easier to understand

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

int main(){
    cout << "This is untidy code.";

    cout <<                     "I'm surprised it works.";

int   number     6;
    cout << "It has" <<
        number <<  "lines of code  - "     ;


    cout << "each more hideous than the last.      ";
                cout << "You probably should add some line breaks in the
text too" << endl;

    cin.get(); //keeps console window open in Visual Studio
return 0;

}
```

Note: Copy and paste from the workbook will cause errors in the IDE

St Ives (Part 1)

The idea here is to translate the following famous riddle to C++. Each number in red needs to be stored in a variable, which you must declare and initialise at the beginning of the program. Then you should use `cout` statements to print the riddle to the console.

As I was going to St. Ives, I met 1 man with 7 wives
Each wife had 7 sacks, each sack had 7 cats, each cat had 7 kits
Kits, cats, sacks and wives,
How many were going to St. Ives?

Hint: Remember the basic structure of a C++ program where you will need to write your code

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

int main(){

    cin.get(); //keeps console window open in Visual Studio
    return 0;
}
```

Note: Copy and paste from the workbook will cause errors in the IDE

St Ives (Part 2)

So we know that the answer to this riddle is just 1. The only person going to St. Ives is the narrator. But how many Kits, Cats, Sacks and Wives did the narrator meet?

Use your variables and perhaps create new ones to calculate and print this total.

Hint: If each wife had 7 sacks, which each contains 7 cats, where each cat had 7 kits.

Then the following pseudocode expression calculates the number of kits.

```
Total Kits   7 (wives) x 7 (sacks) x 7 (cats) x 7 (kits)
```

You will have to do write similar mathematical expressions to find the total number of ‘items’ that the man met when going to St. Ives.

USB Shopper

A girl heads to a computer shop to buy some USB sticks. She loves USB sticks and wants as many as she can get for £50. They are £6 each.

Write a programme that calculates how many USB sticks she can buy and how many pounds she will have left.

You will need some of the following arithmetic operators (e.g. +, -, /, *, %) to achieve this.

Declaration and Initialisation

Set up a new project and follow the below steps:

- Declare a variable with data type integer. Initialise with a value of 8.
- Declare a variable with data type integer. Initialise with a value of 10.
- Declare a third variable of data type integer that sums the answer.
- Print the final variable to console.

Biography

- *Part 1:* Create a program that prints your name, hometown and age to the console. The program has the following constraints:
    - Each item should be stored using a variable of an appropriate data type.
    - Each item should be printed on a new line.
    - You can only use cout once.

- *Part 2:* Modify your program such that the computer asks the user for their name and hometown, then stores these in variables (*hint: we've used cout for output, perhaps there is something similar for input? Also for now only give it your first name).* Now use the variables to print the short biography with the values entered by the user.

Extension Problem (optional):

Try giving the program both your first and second name when asked. What do you notice? Can you provide a fix? Also when asked to enter you age enter a string value? What happens? Can you fix the problem. (*hint:* Google is your friend).

The value of A *(Answer on paper)*

What is the value of A in each of the following code snippets?

a)
```
int a; int b; int c; a   3; b   4; c   5; b   c; a   b;
```

b)
```
int a   3; int b   3; int c   4; c   b; b   a; a   c;
```

c)
```
double a   3.0; double b   6.0; double c  b / a; a   c;
```

d)
```
int a   1; int b   4; a   a * b;
```

e)
```
int a   3; int b   2; a   a % b;
```

Order of Operations *(Answer on paper)*

What is the correct order of operations for the following expressions? How would each be evaluated (e.g. what is the result of each)?

a) `2 + 12 / 4;`

b) `1 + 2   3 + 4 * 5;`

c) `12 % 2   3 + 35 / 5;`

Maths

Given the declarations below, find the result of each expression (Identify the red herring!)

```
int a   3, b   10, c   7;
double x   12.9, y   3.2;
```

a. a + b * c
b. a - b - c
c. a / b
d. b / a
e. a - b / c
f. x / y
g. a + x / b
h. b - x / (y * c)
i. b % a
j. x % y
k. a % b / y

Extension Problem (optional):

Output the results to two decimal places. Google is your friend

---

Temperature

Part 1: Create a program to convert temperature in Fahrenheit to Celsius

*Fahrenheit to celsius → (n - 32) * 0.5556)*

You should make use of variables such that your program prints to console:

```
"X Fahrenheit is Y in Celsius"
```

Where X is the value you specify (as in an appropriate data type) and Y is the calculated result

Part 2: Enhance your program by having the computer ask the user for the temperature in Fahrenheit.

Circles

Write a program that calculates (and displays) the area (A) and circumference (C) of a circle given its radius (r). The program should get the radius from the user. Make your code as clear as possible by using variables of appropriate data types and comments.

$$A = \pi r^2$$
$$C = 2\pi r$$

Bonus Exercise: Paint

Part 1: Start a new project and add the following code. Complete the missing statements so the program functions

```
#include <iostream>
#include <string>
using namespace std;
int main() {
    //consts are variables whose value cannot be modified later
    const int COVERAGE = 350; // paint covers 350 sq ft per gallon
    // declare integers length, width, and height
    // declare double totalSqFt;
    // declare double paintNeeded;
    // prompt for and read in the length of the room
    // prompt for and read in the width of the room
    // prompt for and read in the height of the room
    // Compute the total square feet to be painted - think about   the
dimensions of each wall assuming standard four wall room.
    // Compute the amount of paint needed
    // Print the length, width, and height of the room, the total square
feet and number of gallons of paint required
  cin.get();
```

```
    return 0;
}
```

Part 2: Suppose the room has doors and windows that don't need painting. Ask the user to enter the number of doors and number of windows in the room, and adjust the total square feet to be painted accordingly. Assume that each door is 20 square feet and each window is 15 square feet.

Advanced Note:

When we want to store some information in our program we create a variable. As there name suggests the values stored in a variable can change (they can vary!). If you have a value that shouldn't be altered you can declare this as a constant by using the const keyword at the beginning of the variable declaration. Variables declared as a constant cannot have their value changed.

An Example

Let's say we wanted to create a simple maths program and wanted to store the value of PI in a variable. The value of PI is the same so we don't want this to change, else it could interfere with other calculations in our program. Therefore, it would make sense to declare our PI variable as a constant.

```
const int PIVALUE = 3.14159265359;
```

Constant variable names are typically written in all capital letters to easily identify them as constants in the program.

Chapter 2 - More Operators & IF ELSE

Increment and Decrement Operators

The last chapter introduced some basic operators which can be used to assign values to variables (=) or perform mathematical operations (+, -, /, *, %). There are two further mathematical operators. The increment and decrement operators. These operators increase or decrease the value of a variable for a particular amount.

Here are some examples:

++	(two ‘plus’ signs = increase the value by 1)
--	(two ‘minus’ signs = decrease the value by 1)
+= 2	(‘plus’ sign then ‘equals’ sign then the number 2 = increase the value by 2)
-= 2	(‘minus’ sign then ‘equals’ sign then the number 2 = decrease the value by 2)

In code, increment and decrement operators may be used like this:

```
int time   9;
time ++ ;
cout << time;
```

This code block will print the value 10 to console.

```
int temperature   15;
temperature + 3;
cout << temperature;
```

This code block will print the value 18 to console.

```
int greenBottles   10;
cout << greenBottles + "green bottles sitting on the wall" << endl;
cout << "But if one green bottle should accidentally fall" << endl;
greenBottles  ;
cout << "They'll be " + greenBottles + " green bottles sitting on the wall" <<
endl;
```

This prints to console the first part of the nursery rhyme ’10 green bottles’ using the decrement operator and cout statements.

Relational Operators

In addition to these mathematical and assignment operators we have relational operators which compare two things and either evaluate to TRUE (1) or FALSE (0). This means relational operators can be used to make decisions in computer programs, e.g. if something is true do this, otherwise if its false do that

The following example illustrates the use of the less than relational operator:

```
7 < 10
```

This expression in words is saying “7 is smaller than 10”. This is a TRUE statement.

There are a number of relational operators:

<	(less than)
>	(greater than)
>=	(greater than or equal to)
<=	(less than or equal to)
==	(equality - is the same as)
!=	(inequality - is *not* the same as)

Here are some examples of relational operators in action. Note the use of variables here.

```
int numberOne   6;
int numberTwo   11;
cout << (numberOne    numberTwo);
```

This program will print 0 to console.

```
int numberOne   6;
int numberTwo   11;
cout << (numberOne !  numberTwo);
```

This program will print 1 to console.

Advanced Note

Why doesn’t the console print true or false? In C++ the value of true and false is represented numerically with the values 1 and 0 respectively. Hence why in our first example above 0 (false) is printed to the console, as 6 is equal to 11 is a false statement.

Then in the second example our relational operator asks if the variables are not the same, so 1 (true) is printed to the console, as 6 is not equal to 11 is a true statement.

If you want to print the word rather than 0 or 1 include this line at the top of your main function:

```
cout << boolalpha;
```

Exercises

True or false *(Answer on paper)*

Expressions that include relational operators evaluate to either TRUE (1) or FALSE (0)

Knowing this, state whether the following expressions will evaluate to true or false.

a) `6 <= 10`
b) `'d' != 'd'`
c) `true != false`
d) `int a = 15; int b = 12; cout << (b < a);`
e) `double c = 4.123; double d = 4.123; cout << (d >= c);`
f) `int a = 6; int b = 5; int c = a + b; int d = 10; cout << (d== c);`

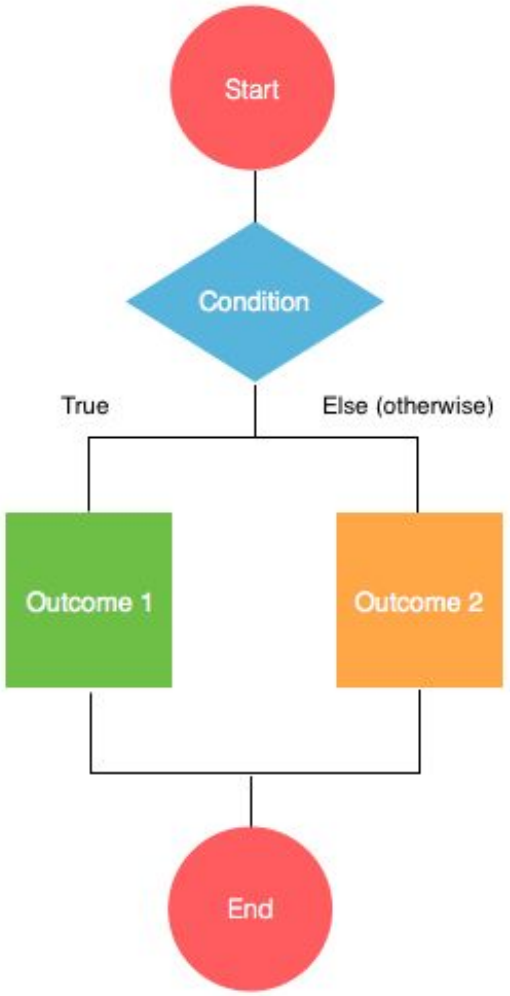
Control Flow - If-Else Statements

The importance of if-else statements

if-else statements are a cornerstone technique in computer programming. They enable us to write ‘decisions’ into our programmes.

It is very possible to make quite complex programmes using just variables, operators and if-else statements.

This is a flowchart diagram for an if-else statement



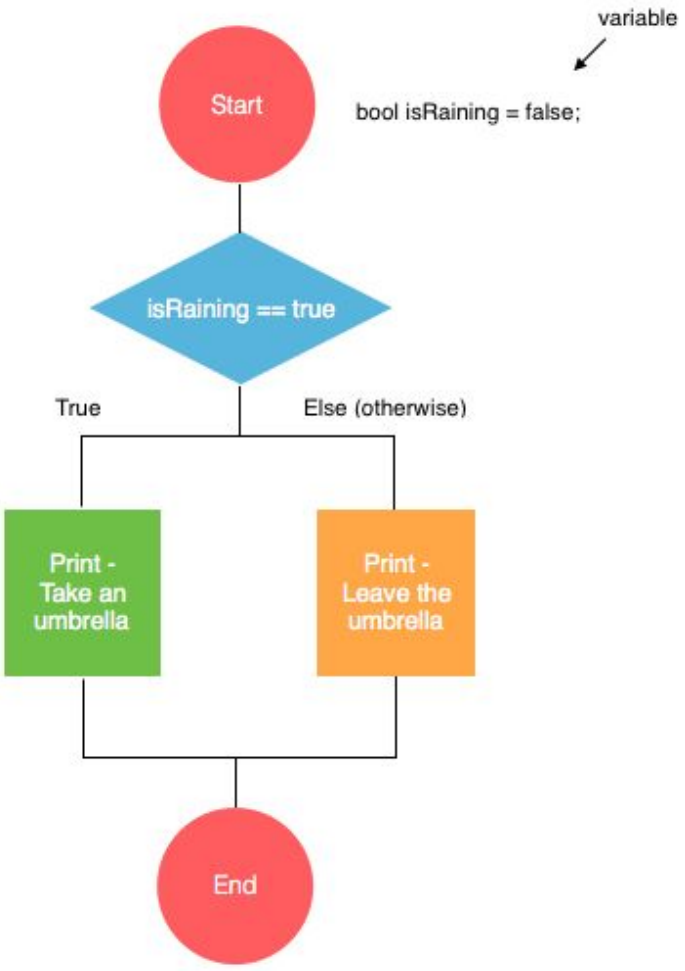
Real world example

This following example evaluates if it is raining. If it is raining, the programme will print “Take an umbrella”, otherwise it will print “Leave the umbrella”.

The condition here is isRaining == true
The code block in green will execute if isRaining == true is a TRUE statement
else (otherwise) the code block in orange will execute.

In this example we can see that the boolean variable isRaining is initialised as false at the beginning of the programme. So, is the condition isRaining == true a TRUE or FALSE statement?

It is FALSE. Therefore the programme will execute the code block in orange and print “Leave the umbrella” to console.



Building blocks of an if-else statement

We can write an if-else statement in pseudocode like this:

```
IF (condition)
   code block to be executed if the condition is true
ELSE
   code block to be executed if the condition is false
END IF
```

The building blocks are therefore:

- A condition: this evaluates as either TRUE or FALSE
- True code block: the code that executes if the condition evaluate to TRUE
- Else code block: the code that executes if the condition evaluates to FALSE

Our real world umbrella example code would therefore look like:

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

int main() {
	bool isRaining   false;

	if(isRaining    true){
		cout << "Take an Umbrella" << endl;
	}else{
		cout << "Leave the Umbrella" << endl;
	}

	return 0;
}
```

Note that the else section is optional and if you only want to run some code if the IF condition is true and do nothing if it is false then you can leave it out.

Important Note: remember code within blocks should be 'indented' to make the if-else statement easier to read.

Checking multiple conditions with ELSE-IF

In the previous examples we have used the else keyword to begin a block of code that will run if our IF condition is false. But what if we wanted to do some additional conditional checks if the first one is false? In this instance we can use the ELSE-IF keyword to create additional conditions.

Take a look at the following example. In this program we want to output a greeting depending on the time of day. If it's before 12 (noon) we want to say "Good Morning". However, we then want more than one other greeting for time after 12 to handle the afternoon and evening. This means we can't just use an ELSE block.

Therefore, we can create a second condition to check if the time is before 6pm (18:00) to handle the afternoon greeting and have a final terminating ELSE block for the evening greeting for all times after 6pm.

An IF statement can have multiple conditional checks by continuing to add ELSE-IF blocks. The program will evaluate the conditions in order and execute the block of code relating to the first condition that evaluates as TRUE, all other blocks will be ignored. If none of the conditions evaluate to TRUE the else block will run, or if no ELSE block is included nothing will happen.

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

int main(){
    int time   8;
    string greeting;

    if (time < 12) {
        greeting   "Good morning";
    } else if (time < 18) {
       greeting   "Good afternoon";
    } else {
       greeting   "Good evening";
    }
    cout << greeting << endl;
}
```

Exercises

It's my Birthday

Look at the following code snippet then describe its operation fully in words (write it down).

```
bool myBirthday   true;
int age   18;

if (myBirthday    true) {
      age++;
       cout << "It is my birthday. I am " << age << " years old";
} else {
       cout << "It is not my birthday" << endl;
}
```

Good Morning?

Write a programme that tells the user whether it is AM or PM. For the purposes of this programme we are using a 24 hour clock.

- Declare a variable called currentTime and initialise it with the value 3.
- Write an if-else statement that compares the currentTime to the value, 12.
- If currentTime is greater than or equal to 12 then print “it is PM” to the console.
- If currentTime is less than 12 then print “it is AM” to the console.

Use the starter code below to complete this exercise, filling in the gaps to complete the solution

```
#include <iostream>
using             std

int main
                  3  //declare and initialise variable for time
    if               //check if 'currentTime' is greater than 12
        cout    "It's PM"    endl  //output PM message
     else  //else 'currentTime' must be less than 12
                             //output AM message

               //keeps console window open on Visual Studio
    return 0
```

Note: Copy and paste from the workbook will cause errors in the IDE

Extension Problem (Optional):

Extend the program to include greetings similar to the example in the “Checking multiple conditions with ELSE-IF” section above

- “Good Morning” for times before 12
- “Good Afternoon” for times before 18
- “Good Evening” for times before 21
- “Good Night” for time before 24
- “Time not Valid” for all other times

Can I vote?

Write a programme that tells the user whether or not they are eligible to vote.

- The programme should declare an appropriate variable with the value of 16.
- It should use an if-else statement to compare this value to the UK voting age, which is 18.
- The programme should print a response to the user for each possible outcome.

Use the starter code below to complete this exercise, filling in the gaps to complete the solution

```
int main
            //declare and initialise age variable
    if          //check if age is greater than or equal to 18
                    //output can vote message
     else  //age must be less that 18
                  //output can vote message

             //keeps console window open on Visual Studio
```

Note: Copy and paste from the workbook will cause errors in the IDE

Splashing out

Write a programme that decides whether or not you can afford to buy the latest computer game. The programme should include:

- A variable that defines the amount of money you have. This should be 40.
- An if-else statements that executes the TRUE code block if the amount of money you have is equal to or above 25.
- The TRUE code block should print “Buy computer game” to the console, and decrement your money variable by 25.
- The ELSE code block should print “I need to save up more cash!”.
- Once the if-else is completed, your programme should print the amount of money you have left over.

Extension Problem (optional):

Allow the user to input how much money they have initially. *Hint: you can utilise cin to get user input*

Odd or Even

Write a program that works out if a number is odd or even

Extension Problem (optional):

- Assign the values of the variable via user input.
- Include appropriate input error checking

Number Checker

Write a program to check whether a number is positive, negative or zero.

Profit or Loss

Write a program to calculate profit or loss. The program should ask the user for the purchase price and sale price then calculate whether profit or loss was made on the sale. For example:

```
Purchase Price: £1250
Sale Price: £1000
Loss = £250
```

Name that Shape

Write a program that determines the name of a shape from its number of sides. Read the number of sides from the user and then report the appropriate name as part of a meaningful message. Your program should support shapes with anywhere from 3 up to (and including) 10 sides. If a number of sides outside of this range is entered then your program should display an appropriate error message.

Temperature

In chapter 1 you wrote a program to convert temperature in Fahrenheit to Celsius.

- Make sure that your program asks the user for the temperature (in Fahrenheit) and prints the correct temperature in Celsius (verify results using an online calculator).
- Extend your program by having the computer ask the user the question shown below (with correct formatting!) and then complete the necessary steps:

Hello, please enter an option:

'1': To convert from Fahrenheit to Celsius

'2': To convert from Celsius to Fahrenheit

- Your program should display an appropriate warning message if the user does not enter a valid integer.

Compound Conditions

Sometimes an if-else statement needs more than one condition. For example, if two things have to be TRUE for some code to run.

For example, what about the scenario where we want to make a cheese sandwich. We need to have both bread and cheese to do this. Without one of the other, it isn't a sandwich.

Logical AND

If we want to programme this scenario, we need a Logical AND operator (&&). Here, we have two conditions in the if-else statement. Both have to evaluate to TRUE for the instruction in the code block to run. In pseudocode, a Logical AND compound condition looks like this:

```
if
        do something;
```

Example: Goldilocks

This Composite If-else Statement is checking for the perfect temperature of porridge: 40 degrees is too cold, and 60 degrees is too hot. The programme prints "That's just right!" when the porridge is more than 40 degrees, but less than 60 degrees.

```
int                         56

if                          40                          60
    cout     "That's just right!"     endl
```

Logical OR

What about if we want an instruction to execute if one condition OR another is TRUE. Well to do this we need a Logical OR operator. We use two 'pipes' for this operator (||)

```
if
      do something;
```

Example: The Bouncer

A nightclub only accepts two forms of ID: A passport OR a Driving Licence. A student who wishes to enter the club has a passport but no driving licence.

```
bool              false
bool                    true

if                  true                          true
      cout    "You may enter the club"    endl
  else
      cout    "Sorry, I can't let you in"    endl
```

You could rewrite this slightly differently. Note the conditions of the IF statement below. You don't need to write hasPassport == true. Instead you can simply write hasPassport.

```
bool              false
bool                    true

if
      cout    "You may enter the club"    endl
  else
      cout    "Sorry, I can't let you in"    endl
```

Logical NOT

The Logical NOT inverts the result of a condition. For example, if you put an exclamation mark in front of TRUE, it becomes FALSE.

Example - Brothers

This example compares the ages of two brothers. If ‘brotherA’ is younger then the system will print “Brother A is older” to the console. Note that the comparison asks whether ‘brotherA’ is younger than ‘brotherB’, and then inverts the answer (i.e. here FALSE becomes TRUE).

```
int             15
int             12
if
       cout     "Brother A is older"     endl
```

Exercises

The Theme Park

At a Theme Park there is a ride that has both a height restriction and an age restriction. To gain admission to the ride you have to be above 0.6 metres tall AND age 5 and above. Write a program that deals with this situation.

You will need a compound condition and the && logical operator to solve this, e.g:

```
if
       //output can ride message
 else
       //output can't ride message
```

Primitive Quiz

You have been asked to write a simple question and answer programme. There is only one question to answer: “What is the capital of France”.

- The programme should ask the user this question, then prompt for a response.
- The user then types in an answer.
- If the answer is correct, the programme should tell the user that their answer was correct.

- If the answer is incorrect, the programme should tell the user that their answer was wrong.

Extension Problem (Optional):

- What about capital letters? (e.g. paris vs. Paris)
- Add additional questions. (e.g. 10 Capitals Cities of Europe Quiz)

Dog Days

It is commonly said that one human year is equivalent to 7 dog years. However this simple conversion fails to recognize that dogs reach adulthood in approximately two years. As a result, some people believe that it is better to count each of the first two human years as 10.5 dog years, and then count each additional human year as 4 dog years.

Write a program that implements the conversion from human years to dog years described in the previous paragraph. Ensure that your program works correctly for conversions of less than two human years and for conversions of two or more human years. Your program should display an appropriate error message if the user enters a negative number.

Letter Checker

In this exercise you will create a program that reads a letter of the alphabet from the user. If the user enters a, e, i, o or u then your program should display a message indicating that the entered letter is a vowel. Any other letter should display a message indicating the entered letter is a consonant

Extension Problem (Optional):

Have your program handle other character inputs - e.g. provide appropriate messages for numbers or special characters like @, !, # etc.

Hint look up "isalpha c++" and "*isdigit c++" on google.*

Mark my Words

You are marking test papers for a local college, and need help keeping track of grade boundaries. For example, you need to be reminded whether 50 - 59% is a C or a D, or if 60 - 69% is a B or an A.
Write a program that does this job for you. Grade Boundaries are as follows:

A - Above 70%
B - 60 - 69%
C - 50 - 59%
D - 40 - 49%
F - Below 40%

Nested if-else statements

A nested if-else statement is one if-else statement wrapped around another. This is useful for when you want to make two decisions in a row, where the second decision relies on the first. In pseudocode the nested if-else statement looks like this:

```
if ( condition1 ) {
       if ( condition 2 ) {
              do something;
       } else {
              do something;
       }
} else {
       do something;
}
```

Note the if-else statement in red is inside (or nested) in the if-else statement in blue.

Example: Zombie Attack!

The world is crumbling following a recent outbreak of the DX TYPE-2 virus. The dead are rising and you must decide whether to fight or flee. You are sheltered in a small cabin in the woods when you hear a knock on the door. You approach the door…

First Question:	Is there a zombie at the door?
Result:	There is a either a zombie at the door or a civilian in need of help.
Second Question:	If there is a zombie at the door you must decide to fight or flee. If there is not you let the civilian in.

We can write the above situation using a nested if-else statement.

```
bool                true
bool                 false
// This is the nested if-else statement that governs the decisions
if                  true
```

```
    if                  true
          cout    "Grab a weapon and fight for your life!"    endl
      else
          cout    "Run away, as fast as you can!"    endl

else
    cout    "Let the civilian in"    endl
```

But remember where a condition == true, we can remove the == true bit to make the code easier to read. So we could rewrite this as:

```
bool               true
bool                false
// This is the nested if-else statement that governs the decisions
if
      if
          cout    "Grab a weapon and fight for your life!"    endl
      else
          cout    "Run away, as fast as you can!"    endl

else
    cout    "Let the civilian in"    endl
```

Exercises

Starting a Band

You want to start a two-piece band, but first you need a friend to join it. If you have a musical friend, you next need to check (using a nested if statement) that they actually play an instrument you want in the band. You decide that they need to play either guitar or drums to join.

Write a program that checks to see if you can start a band! You should use the following variables to write your program:

```
bool               true
string             "guitar"
```

Note: You will need to compare Strings to make this work, remember there are two ways you can do this.

Method 1:

```
string      "Carrot"
string      "Cake"
if
    cout    "Strings match"    endl
  else
    cout    "Strings do not match"    endl
```

Method 2:

```
string      "Carrot"
string      "Cake"
if                    0
    cout    "Strings match"    endl
  else
    cout    "Strings do not match"    endl
```

You will also need to make sure you have included the strings class header at the top of your source file: `#include <string>`

Use the following psuedo-code as a guide to structure your program

```
bool variable
string variable
IF(have musical friend)
      //this if is nested inside the first
      IF(friend plays an instrument)
          output message
      ELSE
          output message
ELSE
    output message
```

Killing Time

You are killing time in the middle of Bath city centre, waiting for your friend who is always late. You receive a text telling you how late they are going to be.

Write a program that follows the below rules:

Scenario 1: Your friend is going to be another 15 minutes or more
Result: If you have more than 3 pounds in change, go buy a coffee, otherwise go for a walk around the town

Scenario 2: Your friend is going to with you in less than 15 minutes
Result: Sit in the park and wait (probably grumbling).

You will need to use a nested if-else statement to complete this task. Please print appropriate text to the console to describe the result of each scenario.

Earthquake

The table below contains earthquake magnitude ranges on the Richter scale and their descriptors. Write a program that reads a magnitude from the user and displays the appropriate descriptor as part of a meaningful message. For example, if the user enters 5.5 then your program should indicate that a magnitude 5.5 earthquake is considered to be a moderate earthquake.

Magnitude	**Descriptor**
Less than 2.0	Micro
2.0 to less than 3.0	Very Minor
3.0 to less than 4.0	Minor
4.0 to less than 5.0	Light
5.0 to less than 6.0	Moderate
6.0 to less than 7.0	Strong
7.0 to less than 8.0	Major
8.0 to less than 10.0	Great
More than 10	Meteoric

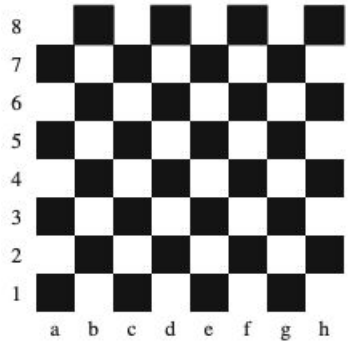
Bonus Exercise: Triangles

Write a program that asks the user to enter the lengths of the three sides of a triangle.

- Use the triangle inequality to determine if we have a triangle: In mathematics, the triangle inequality states that for any triangle, the sum of the lengths of any two sides must be greater than or equal to the length of the remaining side (see here)

If valid, have the program correctly classify the type of triangle as either: Equilateral, Isosceles or Scalene (see here).

Bonus Exercise: Chessboard Check



Positions on a chess board are identified by a letter and a number. The letter identifies the column, while the number identifies the row, as shown above

Write a program that reads a position from the user. Use an if statement to determine if the column begins with a black square or a white square. Then use modular arithmetic to report the color of the square in that row. For example, if the user enters a1 then your program should report that the square is black. If the user enters d5 then your program should report that the square is white. Your program may assume that a valid position will always be entered. It does not need to perform any error checking.

Bonus Exercise: Note Calculator

Write a program that calculates the minimum number of notes required for a given amount of money. Available notes are 500, 100, 50, 20, 10, 5, 2, & 1.

The program should ask the user to enter the amount then calculate the number of notes. For example:

Input

```
Input amount: 575
```

Output

```
Total number of notes

500: 1
100: 0
```

```
50: 1
20: 1
10: 0
5: 1
2: 0
1: 0
```

Chapter 3 - Switch Statements

The Switch Statement

Sometimes we have so many different options in our if-else statements that we need a different approach. This is where the switch statement comes in.

In pseudocode:

```
switch
    case 1
        do something;
        break
    case 2
        do something;
        break
    default
        if other cases aren't true - do this;
```

The switch statement components:

The expression:	This is the variable you are testing against, can be an int or char
Case:	What to do for each possible value of the variable tested
Break:	To show where a case ends
Default:	The case that executes if no other cases match the variable value

Switch statements offer similar functionality to the if else conditional. The switch statement runs through a series of cases and if one of the cases matches the expression then the code block is executed until a break is reached.

Be aware that if a case does not include a break the following case statements will also be executed until a break is reached.

A default case can be included that executes a statement(s) if no case is met (similar to a terminating else). The default usually does not contain a break as this is the last case encountered so it's not required.

The switch statement is limited in that it can only take a single exact value such as int and char values. It cannot take a conditional expression (e.g. `x >= 10`), or string values.

Take a look at the following example to see the switch statement in action

Example: Sugar

This simple example determines whether a user would like to add sugar, the switch statement evaluates the input and outputs the resulting message dependant on the matched case.

```
     << “Would you like sugar?” << endl
char


switch
    case 'Y'
        cout    "Adding sugar..."    endl
         break
    case 'N'
         cout    "No sugar requested..."    endl
         break
    default
         cout    "That input was not recognised"    endl
```

Grouping cases together

If multiple options in your switch statement need to execute the same code, rather than duplicating the lines of code you can group cases together by leaving out the break. For example, if we wanted to account for both upper and lower case inputs in our sugar example we can group together the cases for the upper and lower case letters:

```
     << “Would you like sugar?” << endl
char


switch
    case 'Y'
    case 'y'
        cout    "Adding sugar..."    endl
         break
    case 'N'
    case 'n'
         cout    "No sugar requested..."    endl
         break
    default
         cout    "That input was not recognised"    endl
```

Why use a switch instead of an IF statement?

When you have many options you need to check the switch statement can provide greater clarity and readability of code. It can also offer some minor performance improvements compared to a long IF-ELSE statement.

However, as the switch statement can only accept single exact values in the expression there will be instances where you will have to use an IF-ELSE statement instead (e.g. for evaluating ranges).

Exercises

Continue?

Write a simple program that asks if the user would like to continue playing the game. The program should use a switch statement and accept the values Y (continue) and N (quit). Can you make the program handle both upper and lower case values?

Use the starter code below to complete this exercise, filling in the gaps to complete the solution

```
#include <iostream>
using namespace std
int main
    cout    "Would you like to continue? (Y/N): "  //ask for input
                //variable for user answer;
    cin             //get user input

    switch                  //evaluate expression
        case 'Y'  //case for Y
                        //cout message if user enters Y
                    //break to end case
        case       //case for N
                        //cout message if user enters N
            break
                //default case
                        //default message if neither Y or N entered

            //keeps console window open in Visual Studio
    return 0
```

Note: Copy and paste from the workbook will cause errors in the IDE

Days of the Month

You are a freelance calendar maker. Unfortunately, like me, you are useless are remembering how many days there are in each month of the year.

Write a program that uses a switch statement to tell a user how many days there are in a month.

Your cases should test a number corresponding to the months (e.g. 1 = January, 12 = December), and true cases should print out how many days there are in a month.

Fuel me up

Write a program for an automatic fuel filling service. The program should ask the user for the fuel type using a char (e.g. ‘p’ for petrol or ‘d’ for diesel) and the number of litres needed (int), and finally display the price of the charge.

The program should have the following constraints:

- It should be case sensitive to user input, e.g. "P" and "p" are treated equivalently.
- The program should only display the price if a valid fuel type has been entered and fuel is actually needed!
- You can only use one if-statement and one switch statement.
- If you haven't already, demonstrate how the two statements can be nested without impacting the solution.

Switching Temperature

You should now have a complete working program that converts a given temperature from Fahrenheit to Celsius and vice versa, depending on which option (integer) the user enters.

Revise your temperature converter to use a switch statement. The switch should test a char variable holding the characters entered by the user, e.g: Enter "f" to convert from Fahrenheit to Celsius Enter "c" to convert from Celsius to Fahrenheit

You should provide an appropriate default statement.

Bonus Exercise: Switch Grade Calculator

Using a switch statement write a program that evaluates a student's mark (0-100) to the respective grade. Grade boundaries are defined below

- Less than 40: F
- Between 40 and 50: E
- Between 50 and 60: D
- Between 60 and 70: C
- Between 70 and 80: B
- Greater than 80: A

Your program should ask the user to input the student's full name and mark (between 0 - 100) and then output the student's name and grade (A - F). You should also handle invalid grade values.

Hint: Remember switch statements in C++ can only take constant whole values - e.g. int's and char's. You will need a way of dealing with the numbers between the ranges so they can equal a whole value to use in the switch cases. Think of what the different data types can store.

Chapter 4 - Starting out with loops

What is a Loop?

Loops are used when we want to execute a code statement or several code statements multiple times. We can chose to execute a loop a given number of times, or we can keep looping repeatedly until a certain condition is met.

Why use Loops?

If, for example, we wanted to print all the numbers between 1 and 5 on a new line, we could use `cout` without too much trouble:

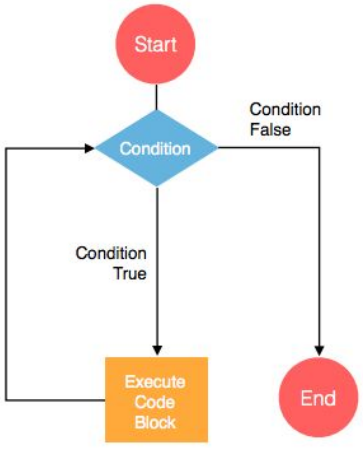
```
cout    1    endl
cout    2    endl
cout    3    endl
cout    4    endl
cout    5    endl
```

But imagine that we wanted to print all numbers between 1 and 1000. One thousand `cout` statements would be incredibly inefficient to programme.

This is where loops come in.

The While Loop

A while loop enables us to loop WHILE a certain condition is true. We can visualise this using the following flow diagram:



The while loop has the following features:

- A Condition: Like the if-else statement the while loop is driven by a condition. As long as that condition evaluates to TRUE, the loop will continue to run
- A Code Block: Statements that will execute if the condition evaluates to TRUE
- A Loop: The code block will execute indefinitely until the condition evaluates to FALSE

We can express a while loop in pseudocode like this:

```
while   condition
```

Back to Counting

So how then can we write a while loop that counts from 1 - 10000, therefore avoiding the messy scenario described above? First we need to declare a variable that will be used to keep track of the current count. We can call it count. Next we create our loop.

```
int          1
while              1000
   cout                  endl
```

Note the features of this while loop:

- The condition: As long as count is less than or equal to 1000 then the loop will continue to run.
- Code Block: This contains two statements. A cout statement that prints to console the current value of count followed by count++, which adds +1 to the value of count each time the loop runs.

The variable count will continue to increment by +1 until it reaches 1001. At that point the condition count <= 1000 will no longer evaluate to TRUE and we exit the loop.

Iterations

An iteration is one cycle of the while loop. So in our counting example there are 1000 iterations.

Exercises

Reverse 9 times table

Using a while loop, and appropriate increment/decrement operators. Write a while loop that prints to console all values of the 9 times table from 108 to 9.

Expected result:

```
108
99
90
81
72
63
54
45
36
27
18
9
```

Use the starter code below to complete this exercise, filling in the gaps to complete the solution

```
#include <iostream>
using namespace std

int main
    int         108  //declare and initalise variable

    while             //while conditional check
        //code to output then decrease number



                  //keeps console window open in Visual Studio
    return 0
```

Note: Copy and paste from the workbook will cause errors in the IDE

The Pointless Box

You are tasked to make an (almost) useless contraption that tells the user when they have entered either the number 1 or 2 to console.

When the user enters 1 the message “you have entered the number 1” should print to console, and when the user enters 2 the message “you have entered the number 2” should print to console.

Hint: This can be solved by having two conditions in the while loop. You will also need an if statement within the loop.

Infinite Loops

When developing the solution to The Pointless Box, you may have encountered a situation where your loop runs indefinitely. This is called an Infinite Loop.

This happens because the condition of the loop always evaluates to TRUE.

Sometimes this is useful - for example when coding for the web when you often want to always be listening for an input from a web server. However in most occasions, we find that we should work to avoid them.

You therefore need to be careful when designing loops that there is a way for the loop to exit or break.

The Break Statement

The break statement forces the loop to cease iterating. When a break statement is executed within a loop, we ‘jump out’ of the loop and continue with the rest of the programme. We saw the break statement in the last chapter on switch statements where its action is the same; When the program encounters a break In a switch it ‘jumps out’ of the statement and continues with the rest of the programme.

Suppose we want to count up to a given number, however that number is defined by user input. In this case we can’t deploy the solution seen in the previous example Why not? There are two reasons:

- We don't know what the user's inputted number is going to be, so setting a condition of e.g. count <= 10000 will likely not stop at the user's chosen number
- This method will only capture numbers up to 10000. The user's number may be higher

```
//this will not work
int            1
while               1000
   cout                  endl
```

One solution to this problem is to use the break statement. It may look something like this:

```
cout     "Enter a whole number: "      endl
int
cin

int            1
while true
  cout                  endl
  if
     break
```

The operation is as follows:

- Cin is used to collect the user's inputted number. This is stored as userNum
- The condition of the while loop is true. This means it will run indefinitely.
- With cout we print the value of our count variable on each iteration.
- We then check to see if our count variable is equal to userNum. If it is we break to loop. Otherwise we increment count and run the iteration again.

Exercises

Eliminate the break

The above example program that outputs all numbers up to the value entered by the user can also be written without the need for the break. Can you modify the solution and eliminate the break?

Brute-Force Attack

You are an inexperienced hacker trying to find the correct passcode to enter a safe. Write a programme that defines the correct passcode of 246, and allows users to enter passcode attempts until they reach the correct number.

Your programme should use a while loop to allow the user to keep entering passcode attempts until correct, and should use a break statement to exit the loop once the correct pass code is entered.

Hint: the condition of your while loop should be true.

Brute-Force Attack II

Modify the solution to Brute-Force Attack to include a maximum of 5 passcode attempts. Each time the user enters an incorrect passcode, they should be prompted of how many passcode attempts remain. If there are 5 failed passcode attempts the while loop should break and inform the user that the authorities have been alerted!

do while Loops

The do while loop is a variant of a while loop, which evaluates the condition at the bottom of the loop instead of the top.

This means that the loop will execute at least once, regardless of whether or not the condition evaluates to TRUE or FALSE.

We can express a do while loop in pseudocode like this:

```
do
        do
  while
```

Note here that the code block to be executed still appears within curly braces. Also note the semicolon at the end of the condition.

Example: Game Music

Game music often loops, often because we don't know how long a player will stay in a particular location. Remember Sonic the Hedgehog (1991)? When you get to the point of fighting Dr Robotnik, boss battle music starts playing. This 30 second sequence will continue to loop until Sonic defeat's him, or is killed.

A do while loop is a valid mechanism to control audio playback:

```
bool                    false
// at this point Sonic enters the battle arena
do
                      true
    cout     "Play battle music"     endl
  while
cout     "Stop battle music"     endl
```

We could improve this implementation by doing two things

- getting rid of redundancies by changing battleRobotnik == true to battleRobotnik
- adding an if statement to exit the loop when the player either dies or defeats Robotnik

```
bool                    false
bool                  false
bool                    false
// at this point Sonic enters the battle arena
do
                    true
  cout     "Play battle music"     endl
  if
                    false

  while
cout     "Stop battle music"     endl
```

Exercises

Input improvement

Copy out the code from do-while example presented in class (see below). Run and test this code. Do you notice any errors while testing it? How might you improve it?

```
#include <iostream>
using namespace std
int main

    char
    do
        cout    "Would you like to Quit (Y/N)?"    endl
        cin
     while            'Y'               'N'

    return 0
```

Note: Copy and paste from the workbook will cause errors in the IDE

Hint: Look up toupper and cin.ignore( )

Loopy

A - Starting with the code below, write a program that counts from zero up to a user specified number. For example, if the user enters the number 5, your program should display the numbers: 0, 1, 2, 3, 4, 5. You cannot use an if-statement for this part!

```
#include <iostream>
using namespace std
int main
   int
   int

            0
```

B - Now have the program start counting from a number input by the user.

C - Use an if-else statement to count the number of even and odd numbers between the first and final value, e.g: start = 1 end = 11 Number of even numbers: 5 Number of odd numbers: 6

D - Write a program that counts and displays numbers starting from the user's first value to their second value (as before), and then count downwards again to the initial value. You can only use one while loop.

Text-based Game

Find the treasure game code in this week's exercises folder in the github reposistory. Copy & paste the code into your IDE.

This is a treasure search game where the user goes through a series of chambers and must find the treasure that is protected by a dragon.

First, follow the flow of the game. It might look difficult to start with but you will soon make sense of it. Try drawing a block diagram of the game to make it easier for you to understand.

Once you understand the operation of the game extend it so that it includes the following features:

1) Add one more chamber to the game after the user meets the dragon.
2) Add more items available for pick up for the user.
3) Add a control that the user can press and get a report of their inventory.

```
E.g.
6 (is pressed)
You have: sword, gloves, lamp etc.
```

Extension problem (Optional):

How might you use your own creativity to expand the game further.

Are you Gullible?

Write a program that continues to asks the user to enter any number other than 5 until the user enters the number 5. If the user enters 5 display the message "Hey! you weren't supposed to enter 5!" and exit the program.

Extension Problem (Optional):

- If the user still hasn't entered the number 5 after 10 iterations display the message "Wow, you're more patient than I am, you win." and exit.

- Instead of always asking the user to enter any number other than 5, instead ask the user to enter any number other than the number equal to the number of times they've been asked to enter a number. (i.e on the first iteration "Please enter any number other than 0" and on the second iteration "Please enter any number other than 1" etc...) The program must still behave accordingly exiting when the user enters the number they were asked not to.

Fuel Extended

Complete the fuel exercise from Chapter 4 ensuring you have used a switch statement and feel free to add other fuel types!

Now extend the program to use a do-while loop that keeps the program running until the user enters a valid fuel type and asks for an appropriate amount of fuel.

Bonus Exercise: Prime number

Create a program that gets a number from the user then checks if the number is a prime number

Bonus Exercise: Exponent

Use a while loop to exponentiate a number. So, for example, $2^4 = 2 \times 2 \times 2 \times 2 = 16$. The number 2 is called the base and the number 4 is called the exponent. Your program should compute the result given positive integers for the base and exponent.

Advanced Note - Error Checking

`cin.fail( );` is a great way of doing some simple error checking. The input stream (`cin`), knows what type of data it is expecting when awaiting user input. If a letter is entered when its expecting an int then an error flag will be placed on the input stream and cin.fail() will evaluate to true. This means you can use `cin.fail( );` as the conditional check in an if statement, or even better a while loop to keep checking user input until valid data is entered.

In the code block after the conditional check you then need to clear the error flag `cin.clear( );` and ignore what is left in the input stream as the invalid data will still be there. `cin.ignore(256, '\n');` tells the input stream to ignore the next 256 characters, or the first new line character it encounters (whichever comes first). As the user will have hit enter when inputting the data it should encounter the new line character first. Now you are safe to ask the user for new data. See the example below:

```
cout    "Enter a number: "    endl
int
cin
while  cin
    cout    "Invalid input"    endl
    cin
    cin        256  '\n'
    cin

cout    "Input number: "         endl
```

Note that the error flag will only occur if the data is of the wrong type (e.g. int when expecting a char, or char when expecting an int. Therefore, `cin.fail( );` won't work for checking if a number is out of range (e.g. 22 when only numbers below 10 are valid). You would need an alternative error checking method for this (e.g. simple if statement check).

Exercises

Age Checker

Write a program that asks the user to input their age. If the user enters incorrect data (e.g. a letter) the program should keep asking them for their age until acceptable data is entered.

Extension Problem (Optional):

Also reject unrealistic ages (e.g. under 0 and over 120).

Sum until fail

Write a program that allows the user to quickly sum a bunch of integers and then displays the result if a digit is not entered. For example:

User enters in console:

```
5
6
3
eggs
```

Program displays:

```
14
```

Notice that the program ignored "eggs".

Chapter 5 - The For Loop

The for Loop

A for loop is very similar to a while loop. The difference is subtle.

You tend to use while loops when you don't know how many iterations of the loop there is going to be. For example, if you are writing a quiz program you might not know how many times a player will get a question wrong.

You use for loops when you do know how many iterations of the loop there need to be.
For example, if you are writing a program that prints every character in word "Computing", you know that you will need to execute the loop 9 times.

while loop vs for loop

The following diagram shows how a while loop that counts from 0 to 10 compares to a for loop that counts from 0 to 10.

```
int count = 0;
while(count <= 10){
    cout << count << endl;
    count++;
}
```

```
for(int count=0; count <= 10; count++){
    cout << count << endl;
}
```

You can see that the while loop and the for loop (in this case) have very similar components. They both contain:

- *Initialisation*: A variable to store the current count. This is the starting point of the loop
  `int 0`
- *Condition*: The condition in which the loop will keep iterating
  `10`
- *Incrementation*: How much the variable count will increase on each iteration of the loop

The difference is that the for loop places all these components within the loops header. These three components are separated by a semicolon when writing the for loop.

Exercises

Some counting

Use your newly acquired knowledge of the for loop to complete the following tasks. Print all values to console in each case.

1. Write a program that counts up from 0 to 50 in increments of 1.
2. Write a program that counts down from 50 to 0 in decrements of 1.
3. Write a program that counts up from 30 to 50 in increments of 1.
4. Write a program that counts down from 50 to 10 in decrements of 2.
5. Write a program that counts up from 100 to 200 in increments of 5.

To get you started here's an example for loop that counts to 1000

```
for int          0           1000
        cout              endl
```

Note: Copy and paste from the workbook will cause errors in the IDE

Odd or Even

Using a for loop, write a programme that prints all values between 20 and 24. Then using an if-else statement and the modulus operator, modify your program to add the word "odd" or "even" to each value to show if it is odd or even.

Expected Output:

```
20 - even
21 - odd
22 - even
23 - odd
24 - even
```

Exercise: Iterate through a word

The following programme uses the `at()` method to print each individual letter of a string (called myWord) to a new console line.

```
string          "Joe"
cout              0     endl
cout              1     endl
cout              2     endl
```

Output:

```
J
o
e
```

Using the `at()` method in combination with a for loop, rewrite the above program to avoid duplicating `cout.` *Note string character counts start at 0 not 1!*

Fibonacci Sequence

This is a harder exercise. For this and the remaining exercise's planning on paper

Watch this video for an explanation of the Fibonacci Sequence:
https://www.youtube.com/watch?v=U2L2nJl8kaw

In short, any number in the Fibonacci Sequence can be calculated by adding up the numbers both one place, and two places before it.

1, 1, 2, 3, 5, 8, 13….

1 + 1 = 2
1 + 2 = 3
2 + 3 = 5
3 + 5 = 8
5 + 8 = 13

and so on…

Using a for loop, write a programme that prints all values of the Fibonacci Sequence between 1 and 55.

Expected Result:

```
1, 1, 2, 3, 5, 8, 13, 21, 34, 55
```

Nested for Loops

A nested loop is a (inner) loop that appears in the loop body of another (outer) loop, much like nested IF statements that we saw in chapter 2.

Example 1: Simple Pattern

The below program prints the following pattern to console:

```
*****
*****
*****
*****
*****
```

```
for  int   1     5          //execute the inner loop 5 times
    for  int   1     5
        cout    "*"    endl    //print 5 stars

    cout    endl  //print to a new console line
```

Above we have one for loop nested within another for loop.

The inner loop prints to the console 5 stars in a row. The outer loop executes the inner loop 5 times, each time printing to a new console line.

Note that we have used the letters **i** and **j** to keep track of what each loops current 'count' is. It is not crucial that you use **i** and **j**, however these variables are typically used in loops.

Example 2: A More Complex Pattern

We can modify the above example so that the following pattern is printed:

```
*****
****
***
**
*
```

```
for  int   1     5          //execute the inner loop 5 times
    for  int         5
         cout    "*"    endl    //print 5 stars

    cout    endl  //print to a new console line
```

The difference between this program and the one seen in Example 1 is subtle. Look at the inner for loop and you will see that the initialisation component is not `int j = 1;` but instead `int j = i;` It utilises the `i` variable from the outer loop. This means that on the first iteration of the loop the inner loop begins at `int j = 1;` but on the second iteration `int j = 2;` and on the third iteration `int j = 3` and so on. Because the starting point for the inner for loop increases by +1 on each iteration of the outer while loop, the number of printed stars will also decrease by -1 on each iteration.

Exercises

Seven Stars, Seven Lines

Write a program that uses nested for loops to print the following pattern to the console

```
*******
*******
*******
*******
*******
*******
*******
```

Exercise: Descending Stars, Seven Lines

Write a program that uses nested for loops to print the following pattern to the console

```
*******
******
*****
****
***
**
*
```

Exercise: Rising Stars, Five Lines

Write a program that uses nested for loops to print the following pattern to the console

```
*
**
***
****
*****
```

Exercise: Rising and Falling Stars

Write a program that uses nested for loops to print the following pattern to the console. Note you will need two sets of nested for loops, one after the other, to achieve this pattern.

```
*
**
***
****
*****
****
***
**
*
```

Exercise - Cubes

Write a program using a for loop to display the cube of the numbers upto an integer entered by the user. Only a single for loop is required for this exercise.

The cube of a number can be calculated by multiplying a number by itself twice (e.g. the cube of 2 is 8 → 2 x 2 x 2).

Expected output if user enters 5:

```
Number is : 1 and the cube of 1 is: 1
Number is : 2 and the cube of 2 is: 8
Number is : 3 and the cube of 3 is: 27
Number is : 4 and the cube of 4 is: 64
Number is : 5 and the cube of 5 is: 125
```

Find the 9s

Write a program to find the number and sum of all integers between 100 and 200 which are divisible by 9. Only a single for loop is required for this exercise.

Bonus Exercise: Multiplication

Using a nested for loop write a program that outputs the 1 - 12 times table.

Sample output:

```
1 times table:
---------------

1 x 1 = 1
1 x 2 = 2
1 x 3 = 3
1 x 4 = 4
1 x 5 = 5
1 x 6 = 6
1 x 7 = 7
1 x 8 = 8
1 x 9 = 9
1 x 10 = 10
1 x 11 = 11
1 x 12 = 12


2 times table:
---------------

2 x 1 = 2
2 x 2 = 4
2 x 3 = 6
2 x 4 = 8
2 x 5 = 10
2 x 6 = 12
2 x 7 = 14
2 x 8 = 16
2 x 9 = 18
2 x 10 = 20
2 x 11 = 22
2 x 12 = 24
```

Bonus Exercise: Reverse it

Write a program that finds the reverse of a number. For example:

```
Input: 12345
Output: 54321
```

Bonus Exercise: For Loop Fixer

```
int      3
for int     0
    cout << "*"
```

Stop this infinite looping by removing or changing just **one** character so that it prints the asterix.

Chapter 6 - Arrays

What is an Array?

An array is a container object that holds a fixed number of values of a single type. While a variable stores one value only, an array stores a series of values. Just like a variable, the array must have a data type, for example int, char, boolean, double etc.

You can think of an array as a single box that contains lots of smaller boxes, where each smaller box holds a single value.

Think of an advent calendar. The advent calendar is a single object that contains multiple numbered compartments that each store chocolate pieces. This can be described as an array.

In this analogy we see the following elements:

- The Array is the advent calendar itself
- Memory locations are the compartments represented by numbered doors
- Data Type stored, which in this case is chocolate

Why use an Array?

Arrays provide a more efficient solution of storing multiple pieces of data than variables.

Take for example this stack of variables that stores five student numbers:

```
int                 323412
int                 373622
int                 387261
int                 318273
int                 362719
```

Now consider we want to store the student number of every individual in Bath Spa University. Using variables alone, we would need over 7000 independent and unique variables! This is unmanageable.

Going back to our five student numbers, we could use one array to store each piece of data. The array could then be represented like this:

323412 373622 387261 318273 362719

Declaring and initialising an Array

The below code shows how to declare an array. An array's dimension is the number of elements it will contain.

```
dataType          dimension
```

Notice the only difference between declaring a normal variable and an array is the addition of square brackets to the end of the name, which are used to declare the array size.

Here is an actual example of an array declaration:

```
int         10
```

This declares an array that can hold 10 integer values.

There are a few components here that we need to address:

- Data Type: The type of data we wish to store (e.g. int, char, String etc)
- The [ ] symbols: This states that what we wish to declare is an array
- Number of Elements: We need to tell the array object how many values we want to store in it. This is so it can allocate computer memory to store the values

Example array declaration and initialisation

Just like we declare and initialise a variable, we need to declare and initialise an array.

A variable: `int        19`

An array: `int       5`

This will create an empty array of integers with 5 placeholders for integers available:

Index (position)	0	1	2	3	4
Value					

NB: The index (position) of elements in an array starts at 0 and not 1 as you might expect

Now that we have declared the array we can add some values at each index.

```
0    19
1    23
2    22
3    30
4    18
```

The Shorthand Method

But hang on, what we have just programmed requires just as many lines of code than storing each value as an independent variable!

Well, we see the benefit when we use shorthand to declare and assign values to positions of the array. In this method C++ will automatically know that the array contains 5 elements so we can choose to omit the dimension value within the square brackets [ ].

```
int           19  23  22  30  18
```

Exercises

Declare and Initialise

1) Declare the following arrays using the correct data type and array size:

a) of characters with 26 elements
b) of integers with 10 elements
c) of strings with 4 elements
d) of decimal numbers with 18 elements

2) Declare and initialise arrays to hold the following information. Don't use the shorthand method just yet

a) 1,2,3,4,5,6,7,8
b) "Hello", "Bye"
c) 'A', 'B', 'C'
d) 1.0, 2.0, 3.0, 4.0, 5.0
e) 10, 20, 30, 40, 50, 60

3) Use the shorthand method to declare and initialise arrays to hold the following information:

a) "", "a", "aa", "aaa"
b) 1,10,100,1000,10000
c) true, false, false, true
d) 5.6, 1.8, 4.34, 7.65

Retrieving values stored within an array

Now we have stored multiple values using an array, we need a way of retrieving them. We do this by referring to the index (or position) of an array.

Here I have created an array (using the shorthand method) to store the resting heart rates of 8 people

```
int            8      54  60  71  59  62  63  60  58
```

Now suppose that I want to print to console the first value (54) and the last value (58)

Firstly, let's remind ourselves what index these two values are positioned at:

54 - index 0
58 - index 7

I could print out the value 54 (index 0) using two approaches.

The first finds the value at index 0 of the heartRates array and stores is to a variable called dataOut. It then prints the value of dataOut to console.

```
int                    0
```

```
cout              endl
```

We could however skip a line of code and print the value at index 0 of the heartRates array directly.

```
cout              0     endl
```

This second approach is better because it removes the need to declare a variable.

Exercises

Print value at index

Consider the following list of vehicles: “Car”, “Train”, “Tractor”, “Skiff”, “Tank”. Declare and initialise an array called vehicles using these values then complete the following tasks:

a) Print index 3 to console
b) Print index 0 to console
c) Print “Tractor Tank” to console using array indexes
d) Print “Car Skiff Train” to console using array indexes

Array Length

To get the size of an array, its length, or in other words the number of elements in the array, we can make use the `sizeof()` function. We can use `sizeof()` on any standard variable and it will return how many bytes the variable takes up in memory (e.g. an integer is 4 bytes). Therefore to get the length of an array we can divide the size of the entire array (the size in bytes of all the elements added together) by one element of the array to get the length. For example, if we had an int array with 10 elements the entire array would be 40 bytes as each individual integer is 4 bytes. So 40 / 4 will return us 10, which is the length of our array. Below is an example of this using an array of type double, where the console will output *8*.

```
double                7.0  7.5  8.0  8.5  9.0  9.5  10.0  10.5
cout    sizeof             sizeof          0      endl
```

Exercises

Array Length

Create a program that outputs the size of arrays containing the following values using the `sizeof()` method described above.

a) 1,2,3,4,5,6,7,8
b) “Hello”, “Bye”
c) ‘A’, ‘B’, ‘C’

Exceptions

An exception in computing programming is an event that disrupts the normal flow of a program.

Copy this array of five elements into your IDE:

```
int           19  23  22  30  18
```

Now try to print the value at index 5:

```
cout        5     endl
```

What happens? In some IDE’s your console indicates an out of bounds exception, whilst in others you might just get a random value. This is because you are trying to access an index that is beyond the scope of the ages array (e.g. a value doesn’t exist at this position).

Adjust the program so that you successfully print the last value of the array, therefore avoiding the exception.

Iterating through an Array

Often a programmer finds that he or she needs to access all the values within an array.

Consider again the array of resting heart rates:

```
int                 54  60  71  59  62  63  60  58
```

If we wanted to print each value to a new console line we could implement this:

```
cout              0     endl
cout              1     endl
cout              2     endl
cout              3     endl
cout              4     endl
cout              5     endl
cout              6     endl
cout              7     endl
```

We however know that when we see repeating code in a computer program it is likely there is a more efficient way of doing things.

What is actually changing in the print statement above? Only the index number, which increments by +1 from index 0 to index 7 across the 8 lines of code.

A for loop provides a more efficient solution.

```
int                 54  60  71  59  62  63  60  58

for  int     0      8
      cout                    endl
```

Remember the for loop includes an initalisation, condition and increment. We can use the initalisation to create a counting variable to run through our array indexes (this variable is typically named i to stand for index). The condition is used to set the limit of our array and the increment adds one to our counting variable each time so we can run through all the array indexes. Inside the for loop body we can then access each array value by using our i counting variable to set the index. As this i variable increases by 1 on each iteration we will be able to access each value.

Work through this for loop one iteration at a time until you fully understand how it works.

Exercises

Heart alteration

Modify the above heart rates example so the for loop condition limit is calculated by the array length rather than being “hard coded”. Check your solution works by adding or removing values from the array.

Hint: Use the `sizeof()` *method within the conditional check*

Iterating through arrays

1. Write a program that uses an array and a for loop to print each letter of the English alphabet.

2. Modify your alphabet loop in the last question to print only the second half of the alphabet.

3. Use a for loop to print all the elements of the following array in reverse order: 1, 3, 5, 7, 9.

4. Write a program that contains an array of integers: 10, 15, 25, 35, 50, 75. Use a for loop and some maths to calculate the average of these values.

5. Write a program that uses an array and a for loop to calculate the average temperature for the past 7 days. Example temperatures: 7.5, 6.3, 10.0, 6.5, 9.1, 11.5, 10.3

6. Write a program that prints out the highest value of a given array. Example values: 6.5, 1.3, 10.9, 7.5, 8.1, 9.9, 9.3

7. Modify your solution to question 6 so that the program also prints the smallest array value.

Write a program that first assigns all numbers between 0 and 100 to an array using a for loop. Using another for loop, iterate through the array and print only the odd numbers.

Working the Array

Write a program that will prompt the user to input ten integer values and store these into an array. Now do the following

- Output all the values in the array after the user has inputted their values
- Print the sum of all values in the array.
- Print the average of the values in the array.

Extension Problem (Optional):

- Make to program print the smallest and greatest numbers in the array.
- Output array in ascending, then descending order

Simple Search

Write a simple search engine that tests if a string is present in a string array. The array you need to declare should be initialised with the following values "Lee" "John", "Sam", "Coral", "Ron", "Jake". The string you are searching for is "Sam"

The program should output all the values in the array then output which position in the array the search term can be found at.

Extension Problem (Optional):

Allow the user to enter the string to be searched for.

Array Counting

Create an array that can hold ten integers, and fill it with numbers input from the user. Display those values on the screen, and then prompt the user for an integer. Go through the array, and count the number of times the item is found, printing the results.

New array

Write a program which takes two arrays of 10 integers each, called a and b. c is an array with 20 integers. The program should put into c the result of appending b onto the end of a, so the first 10 integers of c are from array a, the latter 10 are from b. Then the program should print array c.

Bonus Exercise: Drop it

Create an array holding these numbers: {1, 2, 3, 4, 5, 6}. Write a program that asks for an index and a number. Then it inserts the number at the indicated index of the array, and moves each element after the selected index forward one place (with the last number dropping off the end and disappearing).

Advanced: Multidimensional Arrays

As we have just discussed throughout this chapter an array is a container object that holds a fixed number of values of a single data type. The following code demonstrates how to declare an array for holding 5 values of type int:

```
int         5
```

Recall that an array can be declared and initialized as follows

```
int         5     1, 2, 3, 4, 5
```

Which, in this case saves us from having to assign each element separately:

```
        0    1
        1    2
        2    3
        3    4
        4    5
```

The five elements of myArray are accessed and printed on the console window as follows

```
cout            0     endl
cout            1     endl
cout            2     endl
cout            3     endl
cout            4     endl
cout            5     endl
```

You can think of this as accessing the 5 rows of a single column table:

Value of int at index 0: 1

Value of int at index 1: 2

Value of int at index 2: 3

Value of int at index 3: 4

Value of int at index 4: 5

But what if we want to add another column to our table (array)? This is where multi-dimensional arrays come in!. For example take a look at the following 4 (rows) x 3 (columns) table:

1	5	9
2	6	10
3	7	11
4	8	12

We could use the following 2D (two dimensional) array to represent the table:

```
const int           4
const int           3
int
        0  0    1
        0  1    5
        0  2    9
        1  0    2
        1  1    6
// and so on...
```

Note that the 2D array is setup and accessed using two square brackets [ ] [ ] as opposed to one [ ]. The first set is for the rows and the second is for the columns. In the above case I'm filling one row at a time.

You can also declare and initialize a 2D array.

```
const int           4
const int           3
int
```

```
1  5  9
2  6  10
3  7  11
4  8  12
```

Accessing values in a 2D array is done by specifying the row you want to access in the first square bracket and the column in the second. For example to following would output 10 based on the 2D array declared above: `cout << myArray[1][2] << endl;`

Bonus Exercises

1. Declare and initialise a 2D array to hold the following 5 x 3 table:

2	4	6
3	6	9
4	8	12
5	10	15
5	12	18

2. Use a nested for loop to display the table in the console window, using the same format shown above. You should use `sizeof()` in the loop conditions.

3. Declare - but don't initialise - another 2D array to hold another 5 x 3 table. Fill the arrays using a nested for loop. You should use `sizeof()` in the loop conditions.

4. Writes a program to find the average value of each column of a 2D array of type double

5. Use a string multidimensional array to store the names and respective postcodes of 3 or more people you know. Have the program print out the details of each person on a new line, e.g.

Name	**Postcode**
Bob	A59 1LK

   Modify the program so you get the input from the user.

Further Reading

Arrays are just one type of container in C++. There are many others available that can be extremely useful for handling and manipulating data as your programs become more complex. Two of these (standard library arrays & vectors) we look at in more detail in CodeLab II, but use the resources below if you wish to explore further now.

Standard Library Arrays

C++ (11 onwards) has a library array alongside the built in array.
The library array offers some additional functionality beyond the built in array that can make handling arrays slightly easier.

http://en.cppreference.com/w/cpp/container/array

Vectors

Vectors are similar to arrays in that they store multiple values together as one unit. Unlike arrays however they can be resized and handle their own memory management.

http://www.cplusplus.com/reference/vector/vector/

Standard Template Library

The standard template library contains many other container types including pairs, maps and lists. Each can be extremely useful for handling, modifying and manipulating data.

http://www.cplusplus.com/reference/stl/
https://www.linkedin.com/learning/c-plus-plus-templates-and-the-stl/

Chapter 7 - Functions

What is a Function?

A function is a group of statements that can be bunched together to perform a particular operation. The function can be called again and again as needed.

Say we wanted to perform the following calculation several times in our program (e.g. it might form part of a maths game)

1. Take some integer, inputted by the user
2. Multiply it by 2
3. Add 8
4. Print the result to console

It would be inefficient to write the entire calculation each time we needed to use it. Instead we could create a little algorithm or function that can be called each time we need to execute the calculation.

In pseudocode, the function might look like this:

```
                                2
8
```

Don't worry about the various components for now - we will get to them soon.

We've seen functions before

In CodeLab I we have referred to and used a few functions already. We just may not have realised it. Take a look at the following functions that we use in most workshop sessions:

- the main function: `int main()`
- the compare function from the string class: `compare()`
- `cin.fail( )` to carry out simple error checking.

Function structure

The structure of a function is as follows

```
return_type function_name  parameter list
   //body of the function
```

Return type: specifies the type of data the function should return (e.g. int, bool, double, string etc) back to the main program. If the function does not return anything void can be used as the return type

Function name: Name of the function, this can be anything but usually we specify a name that relates to the task the function performs. E.g. a function named displayMenu would be sensible for a function that displays a menu of options to the user.

Parameter list: When calling a function you can pass it some values that the function can then use to perform its task. We define the parameters a function can be passed in the parameter list. We can specify multiple parameters for use in a function in a comma separated list. For each one we must specify its data type (e.g. int, double, string etc). Parameters are optional, they can be left blank.

Function body: The function body is where we include our set of statements that define the functions task.

The elements of the function structure will become clearer in the forthcoming examples.

Writing a Simple Function

Let's imagine we need to write a function that helps us tune a guitar. We want to know what note each string of the guitar should be tuned to (standard tuning). In this case, it is likely that we will need to print this message a number of times throughout the program.

Expected console output:

```
String 6: E
String 5: A
String 4: D
String 3: G
String 2: B
String 1: E
```

Our function will contain 6 `cout` statements

```
void stringGuide
       cout    "String 6: E"    endl
       cout    "String 5: A"    endl
       cout    "String 4: D"    endl
       cout    "String 3: G"    endl
       cout    "String 2: B"    endl
       cout    "String 1: E"    endl
```

In this example the structure of our function is as follows

Return type: we specify *void* as our function doesn't return a value

Function name: Name of the function, this can be anything but usually we specify a name that relates to the task the function performs. So in this example we use *stringGuide().* For a function that displays a menu we might use the name *displayMenu();*

Parameter list: When calling a function you can pass it some values that the function can then use to perform its task. In this basic example aren't passing in any values so we leave this parameter list blank.

Function body: The function body is where we include our set of statements that define the functions task. So here we have 6 output statements in order to get the expected console output

Invoking Functions

Now that we have written a function for printing the note names associated with each guitar string, we need a way of invoking it. Invoking simply means that we tell the function to run. We might also refer to this as calling the function

We do this by writing the function name with ( ) next to it. So for out string guide function we'd simply write `stringGuide();` in our main function to invoke the function

```
int main
       stringGuide    // invoke the function
      return 0
```

Where do we write functions?

We need to write our function outside of the main() function. If you have multiple functions, you should place them underneath one another in a logical order.

The main function needs to know that a function exists before it is called, otherwise it will throw an error. Therefore you should place functions before the program's main function. You can however declare your function then define it later, this can help keep your code organised and keep your main function towards the top.

Declare vs Define

Declaring a function refers to letting the compiler know that a function exists within the program. When declaring a function we let the compiler know the return type, the name of the function and any parameters the function takes when being invoked / called.

We can then fully define the function later on. Defining a function means we specify the full task the function will perform, so we include the function body with the code statements required for our function to perform its task.

In the `stringGuide();` example above we both declared and defined our function at the same time. Below however, is an example of how we might declare our function ahead of the main, then define the function later on.

```
void stringGuide    //function declaration

int main
       stringGuide    //function call
      return 0


void stringGuide    //function definition
       cout    "String 6: E"    endl
       cout    "String 5: A"    endl
       cout    "String 4: D"    endl
       cout    "String 3: G"    endl
       cout    "String 2: B"    endl
       cout    "String 1: E"    endl
```

Exercises

You say hello, I say goodbye

Write a program that contains and invokes two functions. One that outputs a welcome message that reads “Welcome to my program” and a second that outputs a goodbye message that reads “End of program”.

When completing this exercise first try declaring and defining each function at the same time. Then modify the solution so the functions are declared and defined separately.

Passing in values

To really make use of functions, we need to pass values into them. Remember our earlier *myCalculation()* example. For this function to perform its task we want to pass in an integer number. We can specify the values that a function can take within the parentheses of the function declaration. These are known as the function parameters. These can be named anything you like as long as it’s meaningful.

.

```
void myCalculation int       //function with parameters
             2
             8
      cout           endl
```

Note that you have to declare them with an appropriate data type (in this example an *int*). You can pass as many parameters into your function as you like. Each must be delineated (separated) by a comma.

Invoking functions with arguments

When invoking functions with parameters we can pass in the required data in the parentheses of the function call. With our `myCalculation()` example in the main function I ask the user to enter a number which I then assign to a variable called `userNum`. When invoking the `myCalculation()` function I can pass this userNum variable to the function within the parentheses of the function invocation. This number then gets copied into the `num` variable of the function to perform the calculation.

```
void myCalculation int       //function with parameters
             2
             8
       cout           endl

int main
    int          // create integer variable
    cout    "Enter A Number: "  // ask user to enter a number
    cin             //assign users input to our num variable
                           //invoke the function and pass in userNum
    return 0
```

In this second example we have a function that multiplies two numbers and outputs the result. To do this It uses two numbers that are passed in as the function parameters (*note the two integers specified in the function declaration*). When invoking this function from the main program I can therefore specify the two numbers I want to multiply when invoking the function from the main program.

```
void multiply int       int
       int                      // use passed in parameters
       cout             endl  // output the multiplication result


int main
             12  16   //call the function and pass in two numbers
    return 0
```

Exercises

Passing Parameters

1. Write a function that sums (adds together) three integers. Note that you will need to declare three parameters with the function declaration and invoke the function using three arguments.

2. Write a function that finds the average of three doubles. You will need to pass three doubles into your function, which should sum them and then divide the total by 3.

Returning values from a function

When invoking functions, we may want to return a value back to the main() function. We return values if we want to do something with the returned value.

Let's first of all start with a really simple example. This function returns the current year to the main() function and prints it to console from there:

```
int getYear
    int                2017
    return               // function returns 2017


int main
    int                              // invoke getYear function
    cout                    //output the returned value
    return 0
```

There are few things to note here:

- `void` has been replaced by `int` in the `getYear()` function. Remember this component of a function is called the return type. When we don't return a value from a function we use void, however in this example we want to return an integer, so we replace void with int.

- The returned value is stored as an integer called `returnedValue`. As normal, we need to declare this variable. You can see that `getYear()` is essentially replaced by the value returned by the function - so `getYear()` becomes 2016

A more complex example

In this example we have a function that sums the price of two items. The items have the prices: 25.99, 11.50.

We store the price of the two items as double variables called shoes and tshirt.

The function returns the total value of the items to the main() function. The main function then compares the total to the amount of money we have - 40.00 stored in the variable myMoney - and then prints whether we can afford to buy all three items or not.

```
double sumItems double        double         //declare function

int main
   double           40.00
   double         25.99
   double          11.50

   double                                        //invoke function and pass in
values

   if                            //determine whether can afford items
       cout    "you can afford these items"    endl
     else
       cout    "keep saving up"    endl



double sumItems double        double         //define function
   double                        //calculate cost of items
   return        //return total cost back to main program
```

Observe here that the return type of the function is a `double`. Also note that the total returned from the function `sumItems()` is stored in the variable `returnedTotal` which is then compared to `myMoney` in the if statement.

Exercises

Greetings

Write a program that ask for a name and a greeting, then greets the person. Your function should take two arguments - name and greeting, which will both be strings and then return the new string back to the main program

Can I vote Function

Write a program that asks for the user's name and age, before passing that information into a function called canVote. This function should test if the user is 18 years old or older.

If the user's age is greater than or equal to 18, the function should ask the user for their name.

The function should then return the String “xxx can vote” to the main function, where xxx is the name inputted by the user. If the user’s age is less than 18, the function should return “Too young to vote” to the main function. Finally, print the returned String from the main function.

Go Compare

Write a program that passes two Strings into a function called compareStrings. This function should test whether or not the two Strings have the same value. If the Strings are equal, the function should return the boolean value true to the main function, otherwise it should return the boolean value false. Finally, print the returned value from the main function.

Find the Max

Declare a function called `max_two` that compares the numbers and returns the maximum.

Function Fixer

The following code should calculate and display the mean of two variables of type double. Can you fix the function to make this work?

```
#include <iostream>
using namespace std

void mean  void    void
   return          2.0

int main
   double             5
   double             6
   double
   cout     "a is "
   cout     "b is "
   cout     "The mean of a and b is "
   return 0
```

Say Hello

Write a function titled say_hello() that outputs to the screen "Hello". Modify the function so that it takes an integer argument and says hello the number of times equal to the value passed to it.

Return the product

Write a function that takes two integer arguments and returns an integer that is the product of the two integers. (e.g., integer1: 4, Integer2: 5 returns: 20)

Biography Selection

Write a program that outputs the menu below and gets input from the user before returning the information from a function. You should display the menu via one function called `displayMenu()` and return the information via another function called `bio()`. The program should continue to ask the user for input until they select exit.

```
Pick an option
   1. Show Name
   2. Show Age
   3. Show Hometown
   4. Exit
```

Calculator

Write a program that allows the user to add, subtract, multiply and divide two numbers. The calculations should be performed by function(s) outside of the programs main function. You should present the user with a console menu that allows them to select which calculation they would like to make, then allow them to enter the two numbers before passing these numbers as function arguments to the requested calculation and display the result.

Extension Problem (Optional):

- Allow the user to perform different calculations until they decide to quit the program
- Provide error messages if the user enters invalid data (e.g. letters not numbers).

Bonus Exercise: Power to the Function

Write a function that calculates powers (for example, 4^2 or 6^5). It should ask for a base and an exponent, then return the result to be printed.

Bonus Exercise: Highest Odd

Write a function that returns the highest odd number from a given array. The array is 15, 2, 6, 11, 12, 13, 9

Bonus Exercise: Playing with Strings

Write a program that asks for a users first name and last name separately. The program should pass these strings to a function which returns the users full name as a single string.

Next create another function that replaces every a, e, i , o, u with the letter z and returns the converted string

Create a final function that it reverses the users name and returns the reversed string.

More exercises can be found in the Bonus Exercises workbook