C++ COMPENDIUM

WRITTEN BY

ANRICH TAIT

7 FEBRUARY 2023

Contents

1 SYNOPSIS

This document is a compendium of my C++ learning. Go through each section to find relevant information.

1.1 HOW TO USE THIS DOCUMENT

To use a source code block ensure that C++ and :results output are in the header. Press C-c C-c to project output to the #+RESULTS: block below.

1.2 EXAMPLE TEMPLATE

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

int main() {

    cout << "THIS IS A TEMPLATE";
}
```

Output:

```
THIS IS A TEMPLATE
```

2 THE BASICS

Notes from my early journeys into C++. Here you will find all the basic information you need to get started.

2.1 INTRO

2.1.1 What is C++?

- C++ is a cross-platform language that can be used to create high-performance applications.
- C++ was developed by Bjarne Stroustrup, as an extension to the C language.
- C++ gives programmers a high level of control over system resources and memory.
- The language was updated 4 major times in 2011, 2014, 2017, and 2020 to C++11, C++14, C++17, C++20.

2.1.2 Why Use C++

- C++ is one of the world's most popular programming languages.
- C++ can be found in today's operating systems, Graphical User Interfaces, and embedded systems.
- C++ is an object-oriented programming language which gives a clear structure to programs and allows code to be reused, lowering development costs.
- C++ is portable and can be used to develop applications that can be adapted to multiple platforms.
- C++ is fun and easy to learn!
- As C++ is close to C, C# and Java, it makes it easy for programmers to switch to C++ or vice versa.

2.1.3 Difference between C and C++

- C++ was developed as an extension of C, and both languages have almost the same syntax.
- The main difference between C and C++ is that C++ support classes and objects, while C does not.

2.2 C++ Basics

This section of the document is structured so under each heading there will be an explanation of the relevant topic and then a code block that tests it.

2.3 GENERAL C++ RULES

1. C++ ignores white space, we use white space to make lines more readable.
2. Every statement must end with a semicolon (;).
3.

2.3.1 everyonesfirstprogram.cpp

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

int main() {
    cout << "Hello world!";
    return 0;
}
```

Output:

```
Hello world!
```

1. everyonesfirstprogram explained: Line 1: #include <iostream> is a header file library that lets us work with input and output objects, such as cout (used in line 5). Header files add functionality to C++ programs.

   Line 2: using namespace std means that we can use names for objects and variables from the standard library.

   Line 3: A blank line. C++ ignores white space. But we use it to make the code more readable.

   Line 4: Another thing that always appear in a C++ program, is int main(). This is called a function. Any code inside its curly brackets {} will be executed.

   Line 5: cout (pronounced "see-out") is an object used together with the insertion operator («) to output/print text. In our example it will output "Hello World".

   Line 6: return 0 ends the main function.

   Line 7: Do not forget to add the closing curly bracket } to actually end the main function.

2.4 OMITTING NAMESPACE

You can remove the line **"namespace std;"** and replace it with the **"std"** keyword. See the example below:

```
#include <iostream>

int main(){
    std::cout << "Hello world!";
    return 0;
}
```

Output:

```
Hello world!
```

It is up to you whether or not to include the namespace library (I prefer to).

2.5 COUT (PRINT TEXT)

The **cout** object together with the "«" operator are used to output values.

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

int main() {
    cout << "Hello world!";
    cout << "I am learning C++";
    return 0;
}
```

Take note that the output does not include an empty line between outputs. For that we need to use an escape sequence.

```
Hello world!I am learning C++
```

2.6 ESCAPE SEQUENCES

Escape sequences are used to output nonprintable characters. There are many examples of this, some of the common ones are:

1. $\overset{\frown}{=}$ a horizontal tab
2. \ = a backslash character
3. \" = a double quote charcter
4. = a blank line

2.6.1 BLANK LINES

To insert new lines we need to include the "" character. Two characters after another will instert a blank line.

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

int main() {
    cout << "Hello world! \n\n";
    cout << "I am learning C++";
    return 0;
}
```

Output:

```
Hello world!

I am learning C++
```

2.7 VARIABLES

Variables are containers for storing data values. To create a variable, specify the type and assign it a value.

type variableName = value;

type	variableName	=	value;
int	myNum	=	2000

1. Examples of variables

VARIABLE	USED FOR	EXAMPLE
int	Storing integers (whole numbers)	123 or -123
double	Storing decimal numbers	19.99 or -19.99
char	Storing single characters	’a’ or ’B’
string	Storing text	"Hello World"
bool	Storing values with two states	true or false

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

int main() {
// Declare variables
    int myNum = 5;
    double myFloatNum = 5.99;
    char myLetter = ’A’;
    string myText = "Hello";
    bool myBoolean = true;
// Output variables
    cout << myNum << "\n";
    cout << myFloatNum << "\n";
    cout << myLetter << "\n";
    cout << myText << "\n";
    cout << myBoolean << "\n";
    return 0;
}
```

Output:

```
5
5.99
A
Hello
1
```

2. Adding variables together To add a variable to another you can use the + operator.

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

int main() {
    int x = 5;
    int y = 6;
    int sum = x + y;
    cout << sum;
}
```

Output:

```
11
```

3. Declare multiple variables To declare more than one variable of the **same type** use a comma seperated list.

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

int main() {
    int x = 5, y = 6, z = 50;
    cout << x + y + z;
    }
```

Output:

```
61
```

4. Assign one value to multiple variables You can assign the same value to multiple variables in one line.

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

int main() {
    int x, y ,z;
    x = y = z = 50;
    cout << x + y + z;
    }
```

Output:

```
150
```

2.8 IDENTIFIERS

All variables must be identified with unique names. These are called identifiers. Identifiers can be short (x or y) or more descriptive. Using descriptive names is recommended to create understandable and maintainable code.

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

int main(){
    //Good, easy to see what this variable represents
    int minutesPerHour = 60;
    //Okay, but can get difficult to understand with larger programs
    int m = 60;

    cout << minutesPerHour << "\n";
    cout << m;
    }
```

Output:

```
60
60
```

The general rules for naming variables are:

- Names can contain letters, digits and underscores.
- Names must begin with a letter or an underscore (_).
- Names are case sensitive (myVar and myvar are different variables).
- Names cannot contain whitespaces or special characters like "!#%$,etc".
- Reserved words like (like C++ keywords(int)) cannot be used as names.

2.9 CONSTANTS

When you don't want others(or yourself) to change existing variable values you can use the **const** keyword. This will declare the variable as constant. (unchangeable and read-only). Trying to change a const variable will result in the error: assignment of read-only variable 'myConst'. You should always declare a variable as constant when you have values that are unlikley to change. For example PI or how many minutes in an hour.

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

int main() {
    const int minutesPerHour = 60;
    const float PI = 3.14;

    cout << "There are " << minutesPerHour << "minutes in a hour.\n";
cout << "PI=" << PI;
}
```

Output:

```
There are 60minutes in a hour.
PI=3.14
```

2.10 CIN (USER INPUT)

We have already learnt about outputting with **cout**. The following section explains the use of **cin** to get user input. **cin** is a predefined variable that reads data from the keyboard with the extraction operator "»".

1. Working example of input You can ask Emacs to get the interactive input instead using a named elisp block. Then pass the collected value to the C++ souce block using the :var c-variable=block-name syntax:

   ```
   (completing-read "Type a number: " nil)
   ```

   ```
   #include <stdlib.h>
   #include <iostream>
   using namespace std;

   int main(){
   int myInput = atoi(input);
   cout << "Your number is: " << myInput;
   }
   ```

   Output:

   ```
   Your number is: 10
   ```

   Note that the outputs of source blocks are passed around as strings, so we have to convert it to an integer, hence the atoi and the extra #include.

   This is a super complicated way to retrieve and display input! The following example is an example of normacl C++ code that does this, due to org mode limitations this block does not work.

2. Actual example of input

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

   int main(){
       int myInput;                 //Declare variable
       cout << "Type a number: ";  //Ask user to type a number
       cin << myInput;             // Retrieve user input
       cout << "Your number is: " << myInput; //Display the input value
       }
   ```

2.11 BASIC DATA TYPES

DATA TYPE	SIZE	DESCRIPTION
boolean	1 byte	Stores true or false values
char	1 byte	Stores a single character/letter/number/ASCII value
int	2-4 bytes	Stores whole numbers, without decimals
float	4 bytes	Stores fractional numbers (6-7 decimal digits)
double	8 bytes	Stores fractional numbers (15 decimal digits)

2.11.1 Numeric data types

1. int Use "int" when storing a whole number without decimals.

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

   int main() {
       int myInt = 1000;
       cout << myInt;
       }
   ```

   Output:

   ```
   1000
   ```

2. float Use "float" for storing a number with up to 7 decimals. (floating point number)

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

   int main() {
       float PI = 3.1415926535897;
       cout << PI;
       }
   ```

   Notice that the float was set as 3.1415926535897, which is 15 characters. and the output is exactly 7 characters long.

   ```
   3.14159
   ```

3. double Use "double" for storing a number with up to 15 decimals.

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

   int main() {
       double PI = 3.1415926535897;
       cout << PI;
       }
   ```

Therefore it is safer to use double when dealing with floating point numbers.

```
3.14159
```

4. Scientifc numbers A floating point number can also be a scientific number with an "e" to indicate the power of 10.

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

   int main(){
       float f1 = 35e3;
       double d1 = 12E4;
       cout << "f1 = "<< f1 << "\n";
       cout << "d1 = " << d1;
       }
   ```

   Output:

   ```
   f1 = 35000
   d1 = 120000
   ```

5. Boolean data types A boolean data type is declared with the **bool** keyword and can on take values, true and false. When a value is returned as true = 1 and false = 0

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

   int main() {
       bool isCodingFun = true;
       bool isCodingTasty= false;

       cout << isCodingFun << "\n"; //Outputs true = 1
       cout << isCodingTasty; //Outputs false = 0
       }
   ```

   Output:

   ```
   1
   0
   ```

   Booleans are mostly used for conditional testing, more on this in later sections.

2.11.2 Character data types (char)

The **char** data type is used to store a single character. The character must be surrounded by single quotes, 'a' 'B'

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

int main() {
    char myGrade = 'B';
    cout << myGrade;
    }
```

Output:

```
B
```

Alternatively you can use **ASCII** values to display certain characters.

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

int main() {
    char a = 65, b = 66,c = 67;
    cout << a << b << c;
    }
```

Output:

```
ABC
```

See more ASCII values here: `https://www.cs.cmu.edu/~pattis/15-1XX/common/handouts/ascii.html`

2.11.3 String data types

The **string** data types are used to store a sequence of characters. This type is not built in but but acts like one in it's usage. String values must be surrounded by double quotes ("")

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

int main() {
    string greeting = "Hello, this is how to use a string data type!";
    cout << greeting;
    }
```

Note: to use the string data type you need to include the header file **#include <string>** to make use of the string library.

```
Hello, this is how to use a string data type!
```

2.12 OPERATORS

Operators are used to perform operations on variables and values. It can also be used to add together a variable and a value or a variable and another variable. C++ divides operators into the following categories:

2.12.1 Arithmetic operators

Arithmetic operators are used to perform mathematical operations.

OPERATOR	NAME	DESCRIPTION	EXAMPLE
+	Addition	Adds together two values	x + y
-	Subtraction	Subtracts two values	x - y
*	Multiplication	Multiplies two values	x * y
%	Modulus	Returns the division remainder	x % y
++	Increment	Increases the value of a variable by 1	++x
–	Decrement	Decreases the value of a variable by 1	–x

Modulus is kind of obscure, here is an example: The modulus operator (also informally known as the remainder operator) is an operator that returns the remainder after doing an integer division. For example, 7 / 4 = 1 remainder 3. Therefore, 7 % 4 = 3. As another example, 25 / 7 = 3 remainder 4, thus 25 % 7 = 4.

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

int main () {
    int myAdd = 5 + 4;      //addition
    int mySub = 5 - 4;      //subtraction
    int myMul = 5 * 4;     //multiplication
    int myDiv = 5 / 4;   // division
    int myMod = 5 % 4;      //modulus
//    int myInc = ++5 + 1;        //increment (I don't understand increment yet so this wi
//    int myDec = --5 - 1;        //decrement ( """" )

    cout << "Addition: "<< myAdd << "\n";
    cout << "Subtraction: "<< mySub << "\n";
    cout << "Multiplication:" << myMul << "\n";
    cout << "Division:" << myDiv << "\n";
    cout << "Modulus" << myMod << "\n";
//    cout << myInc << "\n";
//    cout << myDec;
    }
```

Output:

```
Addition: 9
Subtraction: 1
Multiplication:20
```

```
Division:1
Modulus1
```

1. **TODO** Learn about decrement and increment!

2.12.2 Assignment operators

Assignment operators are used to assign values to variables.

1. Table of assignment operators:

OPERATOR	EXAMPLE	SAME AS
=	x = 5	x = x
+=	x+= 5	x = x + 5
-=	x-= 5	x = x - 5
*=	x*= 5	x = x * 5
/=	x/= 5	x = x / 5
%=	x%= 5	x = x % 5
&=	x&= 5	x = x & 5
ˆ=	xˆ= 5	x = x ˆ 5
»=	x»= 5	x = x » 5
«=	x«= 5	x = x « 5

2. Example of each:

   (a) =

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

   int main() {
       int x = 5;
       cout << x;
       return 0;
       }
   ```

   Output:

   ```
   5
   ```

   (b) +=

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

   int main() {
       int x = 5;
       x += 3;
       cout << x;
       return 0;
       }
   ```

   Output:

```
8
```

(c) -=

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

int main() {
    int x = 5;
    x -= 3;
    cout << x;
    return 0;
    }
```

Output:

```
2
```

(d) "*="

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

int main() {
    int x = 5;
    x *= 3;
    cout << x;
    return 0;
    }
```

Output:

```
15
```

(e) /=

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

int main () {
    double x = 5;
    x /= 3;
    cout << x;
    return 0;
    }
```

Output:

```
1.66667
```

(f) %=

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

int main() {
    int x = 5;
    x %= 3;
    cout << x;
    return 0;
    }
```

Output:

```
2
```

(g) &=

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

int main() {
    int x = 5;
    x &= 3;
    cout << x;
    return 0;
    }
```

Output:

```
1
```

(h) »=

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

int main() {
    int x = 5;
    x >>= 3;
    cout << x;
    return 0;
    }
```

Output:

```
0
```

(i) «=

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

int main() {
    int x = 5;
    x <<= 3;
    cout << x;
    return 0;
    }
```

Output:

```
40
```

2.12.3 Comparison operators

Comparison operators are used to compare two values (or variables). This is important for programming becuase it helps us find answers and make decisions.

The return value of a comparison is either 1 or 0, which means true or false. These values are known as boolean values. More about this in the BOOLEANS and IF...ELSE sections. In the following example > is used to find out if 5 is greater than 3.

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

int main() {
    int x = 5;
    int y = 3;
    cout << (x > y); //returns 1 (true) because 5 is greater than 3
    }
```

Output:

```
1
```

1. Table of comparison operators:

OPERATOR	NAME	EXAMPLE
==	Equal to	x == y
!=	Not equal	x != y
>	Greater than	x > y
<	Less than	x < y
>=	Greater than equal to	x >= y
<=	Less than or equal to	x <= y

2.12.4 Logical operators

As with comparison operators you can also test for true(1) or false(0) values with logical operators.

Logical operators are used to determine the logic between variables or values. Types of logic operators:

- && (logical and) = Returns true if both statements are true.
- || (logical or) = Returns true if one of the statements is true.
- ! (logical not) = Reverse the result, returns false if the result is true.

2.12.5 Bitwise operators (more on this in later sections)

3 ALL YOU NEED TO KNOW ABOUT STRINGS

- Strings are used for storing text.
- To use a string you must include an additional header file in the source code, namely the #include <string> library.
- A **string** variable contains a collection of characters surrounded by double quotes

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

int main() {
    string reminder = "Remember to include the <string> library!";
    cout << reminder;
    return 0;
}
```

Output:

```
Remember to include the <string> library!
```

3.1 STRING CONCATENATION

- The "+" operator can be used between strings to add them together to make a new string. This is called **concatenation**.
- C++ uses the "+ " operator for both addition and concatenation. If you attempt to add a number to a string you will output an error.

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

int main() {
    string firstName = "Anrich";
    string lastName = "Tait";
    string fullName = firstName + " " + lastName; //see the use of inverted commas to crea
    cout << fullName;
    return 0;
    }
```

Output:

```
Anrich Tait
```

3.1.1 Append

A string is actually an object which can contain functions that perform certain operations. For example the append function.

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

int main() {
    string firstName = "Anrich";
    string lastName = "Tait";
    string fullName = firstName.append(lastName);
    cout << fullName;
    return 0;
    }
```

Output:

```
AnrichTait
```

Notive this doesn't insert a space for you between variables.

3.2 STRING LENGTH

To get the length of a string use the "length()" or "size()" function.

- The "size()" and "length()" function are aliases for one another, therefore it is probably best to use the "size()" function (for optimization purposes of course :).

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

int main() {
    string alphabet = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
    cout << "The length of the alphabet is: " << alphabet.size();
    return 0;
    }
```

Output:

```
The length of the alphabet is: 26
```

3.3 ACCESS STRINGS

You can access the characters in a string by referring to its index number inside square brackets.

- String indexes start with 0 [0] as the the first character and 1 [1] as the second etc.

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

    int main() {
        string alphabet = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
        cout << alphabet[0] << "\n"; // outputs A
        cout << alphabet[1]; // outputs B
        return 0;
        }
```

Output:

```
A
B
```

3.3.1 Change string characters:

To change the value of a specific character in a string, refer to it's index number and use single quotes.

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

int main() {
    string myString = "Hello";
    myString[0] = 'J';
    cout << myString;
    }
```

Output:

```
Jello
```

3.4 STRINGS - SPECIAL CHARACTERS

In the below example it shows an attempt at outputting a string with quotes.

string txt = "We are the so-called **"Vikings"** from the north!"

This will output an error relating to the use of quotes inside a string. The correct way to do this is to use special characters like mentioned in the basics section.

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

int main() {
    string txt = "We are the so-called \"Vikings\" from the north!";
    cout << txt;
    return 0;
    }
```

Output:

```
We are the so-called "Vikings" from the north!
```

3.5 USER INPUT STRINGS

- It is possible to use the extraction (input) operator "»" on cin to a display a string entered by a user.
- However using **"cin"** considers whitespace as a terminating character, which means that it will only output a single word.
- For this reason it's better to use the **"getline"** function to read a line of text.

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

    int main() {
        string fullName;
        cout << "Type your full name: ";
        getline (cin, fullName);
        cout << "Your name is: " << fullName;
        }
```

The getline() function will take "cin" as the first parameter and the string variable as the second.

4 MATH

To include math related functions in your code you need to include the <math> library.

- In this library there are many math functions, the most popular are:

FUNCTION	DECRIPTION
max	find the highest value of x and y
min	find the lowest value of x and y
abs(x)	returns the absolute value of x
acos(x)	returns the arccosine of x
asin(x)	returns the arcsine of x
atan(x)	returns the arctangent of x
cbrt(x)	returns the cube root of x
ceil(x)	returns the value of x rounded up to it's nearest integer
cos(x)	returns the cosin of x
cosh(x)	returns the hyperbolix cosine of x
exp(x)	returns the value Ex
expm1(x)	returns ex-1
fabs(x)	returns the absolute value of floating x
fdim(x,y)	returns the postitive difference between x and y
floor(x)	returns the value of x rounded down to its nearest integer
hypot(x,y)	returns sqrt(x2+y2) without intermediate overflow or underflow
fma(x,y,z)	returns x*y+z without losing precision
fmax(x, y)	returns the highest value of a floating x and y
fmin(x, y)	returns the lowest value of a floating x and y
fmod(x, y)	returns the floating point remainder of x/y
pow(x, y)	returns the value of x to the power of y
sin(x)	returns the sine of x (x is in the radians)
sinh(x)	returns the hyperbolic sine of a double value
tan(x)	returns the tangent of an angle
tanh(x)	returns the hyperbolic tangent of a double value

5 BOOLEANS

When you need a data type that can only have one of two values like:

- YES / NO
- ON / OFF
- TRUE / FALSE

Then **bool** is your go to. Due to it either outputting 1 for **true** or 0 for **false**.

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

int main() {
    bool isCodingFun = true;
    bool isCodingTasty = false;
    cout << isCodingFun;
    cout << isCodingTasty;
    return 0;
    }
```

Output:

```
10
```

In the above example you can see that true returns 1 and false returns 0. However it is more common to return a boolean value by comparing values and variables.

5.1 BOOLEAN EXPRESSIONS

A **boolean expression** returns a boolean value that is either 1 or 0. This is useful in cases to build logic and find answers. Using comparison operators is the name of the game with regards to booleans.

First look at a simple example then see how this concept can be applied to real life.

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

int main() {
    cout << (10 > 9);
    }
```

Output:

```
1
```

Returns 1 (true) because 10 is in fact higher than 9.

In the following example we use the >= operator to find out if the age (25) is greater than or equal to voting age limit, which is pre-set to 18. It makes use of a simple **if. . . else** expression to determine output. (MORE ON IF. . . ELSE IN FURTHER SECTIONS!)

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

int main() {
    int citizenOneAge = 25; //Citizen one's age
    int citizenTwoAge = 17; //Citizen two's age
    int votingAge = 18;     //This is the pre-set voting age

    if (citizenOneAge >= votingAge) {
cout << "You are old enough to vote! \n";
    }   else {
cout << "You are not old enough to vote. \n";
    }

     if (citizenTwoAge >= votingAge) {
cout << "You are old enough to vote! \n";
    }   else {
cout << "You are not old enough to vote. \n";
   }
    return 0;
    }
```

Output:

```
You are old enough to vote!
You are not old enough to vote.
```

So we can see in this example that citizen one is old enough to vote, while citizen two is not. Booleans are the basis for all C++ comparisons and conditions.

6 CONDITIONS (IF...ELSE)

In C++ you can use logical conditions from mathematics to perform different actions for different decisions. C++ has the following conditional statements:

STATEMENT	USED
if	to specify a block of code to be executed if a specified condition is true
else	to specify a block of code to be exectued if the same condition is false
else if	to specify a new condition to test if the first condition is false
switch	tp specify many alternatives blocks of code to be executed

6.1 THE if STATEMENT

Use the **if** statement to specify a block of code to be executed if a condition is true.

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

int main() {
// if (condition) {
//      block of code to be executed if condition is true
//  }

    int x = 20;
    int y = 18;
    if (x > y) {
cout << "x is greater than y";
}
    return 0;
    }
```

Output:

```
x is greater than y
```

6.2 The else STATEMENT

Use the **else** statement to specify a block of code to be executed if a condition is false/

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

int main() {
/*
    if (condition) {
(block of code to be executed if true)
     } else {
```

```
(block of code to be executed if false)
     }
*/

    int time = 20;
    if (time < 18) {
    cout << "Good day.";
    } else {
    cout << "Good evening.";
    }
    }
```

Output:

```
Good evening.
```

In the above example time(20) is greater than 18 so the condition is **false**. Because of this we move on to the **else** condition and output "Good evening." If the time was less than 18, the program would print "Good day.".

6.3 THE else if STATEMENT

Use the **"else if"** statment to specify a new condition if the first condition is **false**.

6.3.1 SYNTAX:

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

int main() {
    if (condition1) {
// block of code to be executed if condition1 is true
    } else if (condition2) {
// block of code to be executed if the condition1 is false and condition2 is true
    } else {
// block of code to be executed if conditon1 is false and condition2 is false
    }
    }
```

6.3.2 EXAMPLE:

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

int main() {
    int time = 22;
    if (time < 10) {
cout << "Good morning.";
    } else if (time < 20) {
```

```
cout << "Good day.";
    } else {
cout << "Good evening.";
    }
    }
```

Output:

```
Good evening.
```

In the above example time(22) is greater than 10 (morning), so the first condition is false. The next condition in the **else if** statement is also false so the compiler moves on to the **else** condition, since condition1 and condition2 are both false. This then outputs "Good evening." If the time was 14 it would output "Good day.".

6.4 SHORT HAND if else (TERNARY OPERATOR)

The ternary operator consists of three operands. It can be used to replace multiple lines of code with a single line. It is often used to replace simple if else statements.

6.4.1 SYNTAX:

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

int main() {
    variable = (condition) ?
    expressionTrue: expressionFalse;
    }
```

6.4.2 EXAMPLE:

Instead of writing:

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

int main() {
    int time = 20;
    if (time < 18) {
cout << "Good day.";
    } else {
cout << "Good evening.";
    }
    }
```

Output:

```
Good evening.
```

We can write:

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

int main() {
    int time = 20;
    string result = (time < 18) ? "Good day." : "Good evening.";
    cout << result;
    }
```

Output:

```
Good evening.
```

7 SWITCH STATEMENTS

Use the "**switch**" statement to select one of many code blocks to be executed.

7.1 SYNTAX:

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

int main() {
    switch(expression) {
case x:
    //code block
    break;
case y:
    //code block
    break;
default:
    //code block
    }
    }
```

7.1.1 HOW IT WORKS:

- The **"switch"** expresiion is evaluated once.
- The value of the expression is compared with the values of each "**case**".
- If there is a match, the associated block of code is executed.
- The "**break**" and "**default**" keywords are optional, and will be described later in the section.

7.2 EXAMPLE:

This example uses the weekday number to calculate the weekday number.

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

int main() {
    int day = 4;
    switch (day) {
case 1:
    cout << "Monday";
    break;
case 2:
    cout << "Tuesday";
    break;
case 3:
    cout << "Wednesday";
    break;
case 4:
    cout << "Thursday";
    break;
case 5:
    cout << "Friday";
    break;
case 6:
    cout << "Saturday";
    break;
case 7:
    cout << "Sunday;";
    break;
    }
    }
```

Output:

```
Thursday
```

7.3 THE BREAK KEYWORD

When C++ reaches a "**break**" keyword, it breaks out of the switch block. This will stop the execution of more code and case testing inside the block. When a match is found and the job is done it's time for a break. There is no need for more testing. A break can save a lot of execution time because it "ignores" the exectution of all the rest of the code in the switch block.

7.4 THE DEFAULT KEYWORD

The "**default**" keyword specifies some code to run if there is no case match.

7.4.1 EXAMPLE:

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

int main() {
    int day = 4;
    switch (day) {
case 6:
    cout << "Today is Saturday";
    break;
case 7:
    cout << "Today is Sunday";
    break;
    default:
cout << "Looking forward to the Weekend";
    }
}
```

Output:

```
Looking forward to the Weekend
```

8 WHILE LOOP

8.1 LOOPS

Loops can execute a block of code as long as a specified condition is reached. Loops are handy because they save time, reduce errors and they make code more readable.

8.2 WHILE LOOPS

The "**while**" loop loops through a block of code as long as a specified condition is true.

8.2.1 SYNTAX:

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

int main() {
    while (condition) {
//code block to be executed
    }
    }
```

8.2.2 EXAMPLE:

In the example, the code in the loop will run over and over again as long as a variable (i) is less than 5:

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

int main() {
    int i = 0;
    while (i < 5) {
    cout << i << "\n";
    i++;
    }
    }
```

Output:

```
0
1
2
3
4
```

Note: Do not forget to increase the variable used in the condition, otherwise the loop will never end!

8.3 DO/WHILE LOOPS

The "**do/while**" loop is a variant of the "**while**" loop. This loop will execute the code block once, before checking if the condition is true, then it will repeat the loop as long as the condition is true.

8.3.1 SYNTAX

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

int main() {
    do {
//code block to be executed
    }
    while (condition);
    }
```

8.3.2 EXAMPLE:

This example uses a "**do/while**" loop. The loop will always be executed at least once, even if the condition is false, because the code block is executed before the condition is tested.

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

int main() {
    int i = 0;
    do {
    cout << i << "\n";
    i++;
    }
    while (i < 5);
    }
```

Output:

```
0
1
2
3
4
```

Note: Do not forget to increase the variable used in the condition, otherwise the loop will never end!

9 FOR LOOP

When you know exactly how many times you want to loop through a block of code you can use the "**for**" loop instead of the "**while**" loop.

9.1 SYNTAX

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

int main() {
    for (statement 1; statement 2; statement 3;) {
//block of code to be executed
    }
    }
```

1. Statement 1 is executed (one time) before the execution of the code block.
2. Statement 2 defines the condition for executing the code block.
3. Statement 3 is executed (every time) after the code block has been executed.

9.2 EXAMPLE

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

int main() {
    for (int i = 0; i < 5; i++) {
cout << i << "\n";
    }
    }
```

Output:

```
0
1
2
3
4
```

1. Statement 1 sets a variable before the loop starts (int i = 0).
2. Statement 2 defines the condition for the loop to run (i must be less than 5). If the condition is true, the loop will start over again, if it's false the loop will end.
3. Statement 3 increases a value (i++) each time the code block in the loop has been executed.

9.3 EXAMPLE 2

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

int main(){
    for (int i = 0; i <= 10; i = i + 2) {
cout << i << "\n";
    }
    }
```

Output:

```
0
2
4
6
8
10
```

This example prints only the even values between 0 and 2 (see "i + 2").

10 NESTED LOOPS

It is also possible to place a loop inside another loop. This is called a "**nested loop**". The "inner loop" will be executed one time for each iteration of the "outer loop".

10.1 EXAMPLE

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

int main() {
    //Outer loop
    for (int i = 1; i <= 2; ++i) {
cout << "Outer: " << i << "\n";
    //Executes 2 times

    //Inner loop
    for (int j = 1; j <= 3; ++j) {
cout << "Inner: " << j << "\n";
    //Executes 6 times (2 * 3)
    }
    }
    }
```

Output:

```
Outer: 1
Inner: 1
Inner: 2
Inner: 3
Outer: 2
Inner: 1
Inner: 2
Inner: 3
```

11 FOREACH LOOPS

There is also a "**for-each** loop" (introduced in C++ verion 11(2011)) which is used exclusively to loop through elements in an array (or other data sets).

11.1 SYNTAX

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

int main() {
    for (type variableName : arrayName)
{
//codeblock to be executed
}
    }
```

11.2 EXAMPLE

The following example outputs all elements in an array using a "for-each loop":

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

int main() {
    int myNumbers[5] = {10, 20, 30, 40, 50};
    for (int i : myNumbers) {
cout << i << "\n";
    }
    }
```

Output:

```
10
20
30
40
50
```

12 BREAK AND CONTINUE

12.1 BREAK

In a previous section the "**break**" statement was used to "jump out" of a switch statement. The "**break**" statement can also be used to jump out of a loop. The following example shows how break is used to jump out of a loop when i is equal to 4:

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

int main(){
    for (int i = 0; i < 10; i++) {
if (i == 4) {
break;
}
    cout << i << "\n";
    }
    }
```

Output:

```
0
1
2
3
```

12.2 CONTINUE

The "**continue**" statement breaks one iteration (in the loop) if a specified condition occurs and continues with the next iteration in the loop. This example skips the value of 4.

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

int main() {
    for (int i = 0; i < 10; i++) {
if (i == 4) {
continue;
}
cout << i << "\n";
    }
    }
```

12.2.1 BREAK AND CONTINUE IN WHILE LOOPS

You can also use "**break**" and "**continue**" in while loops:

1. BREAK EXAMPLE

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

int main() {
    int i = 0;
    while ( i < 10) {
cout << i << "\n";
i++;
if (i == 4) {
    break;
}
    }
    }
```

Output:

```
0
1
2
3
```

2. CONTINUE EXAMPLE

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

int main() {
    int i = 0;
    while (i < 10) {
if (i == 4) {
    i++;
    continue;
}
cout << i << "\n";
i++;
    }
    }
```

Output:

```
0
1
2
3
5
6
7
8
9
```

13 ARRAYS

Arrays are used to store mulitple values in a single variable, instead of declaring seperate variables for each value. To declare an array, define the variable type, specify the name of the array followed by square brackets and specify the number of elements it should store.

```
string cars[4];
```

We have no declared a variable that holds an array of four strings. To insert values to it we can use an array literal - place the values in a comma-seperated list, inside curly brackets:

```
string cars[4] = {"Volvo", "BMW", "Audi", "Mclaren"};
```

To create an array of 3 integeres it's much the same:

```
int myNum[3] = {1, 2, 3};
```

13.1 ACCESS THE ELEMENTS OF AN ARRAY

You can access an array element by referring to the index number inside square brackets []. The following statement accesses the value of the first element in cars:

```
string cars[4] = {"Volvo", "BMW", "Audi", "Mclaren"};
cout << cars[0]; // outputs volvo
cout << "\n" << cars[3]; // outputs mclaren
```

13.2 CHANGE AN ARRAY ELEMENT

To change the value of a specific element, refer to the index number:

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

int main() {
    string cars[4] = {"Volvo", "BMW", "Audi", "Mclaren"};
    cars[0] = "Mercedes";
    cout << cars[0];
    }
```

Output:

```
Mercedes
```

14 ARRAYS AND LOOPS

14.1 LOOP THROUGH AN ARRAY

You can loop through the array elements with the "**for**" loop. The following example outputs all elements in the "**cars**" array:

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

int main() {
    string cars[5] = {"Volvo", "BMW", "Audi", "Mclaren", "Tesla"};
    for (int i = 0; i < 5; i++) {
cout << cars[i] << "\n";
    }
    }
```

Output:

```
Volvo
BMW
Audi
Mclaren
Tesla
```

The following example outputs the index of each element together with it's value.

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

int main() {
    string cars[5] = {"Volvo", "BMW", "Audi", "Mclaren", "Tesla"};
    for (int i = 0; i < 5; i++) {
cout << i << " = " << cars[i] << "\n";
    }
    }
```

Output:

```
0 = Volvo
1 = BMW
2 = Audi
3 = Mclaren
4 = Tesla
```

The following example shows how to loop through an array of integers.

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

```
int main() {
    int myNumbers[5] = {10, 20, 30, 40, 50};
    for (int i = 0; i < 5; i++) {
cout << myNumbers[i] << "\n";
    }
    }
```

Output:

```
10
20
30
40
50
```

14.2 THE foreach LOOP

This loop is used exclusively to loop through elements in an array:

14.2.1 SYNTAX

```
for (type variableName : arrayName)
    {
    //code block to be executed
    }
```

14.2.2 EXAMPLE

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

int main() {
    int myNumbers[5] = {10, 20, 30, 40, 50};
    for (int i : myNumbers) {
cout << i << "\n";
    }
    }
```

Output:

```
10
20
30
40
50
```

14.3 OMIT ARRAY SIZE

In c++ you don't have to specify the size of the array. The compiler is smart enough to determine the size of the array based on the number of inserted values.

15 TEST BLOCK FOR GNU EMACS

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

int main() {
  cout << "test";
}
```