Complete notes on

C++

Written By:-
Dipti Gaydhane
(Software Engineer)

Page No.
Date

INDEX

C++ TUTORIAL

Page No.
Date

12) C++ MATH

13) C++ BOOLEANS

14) C++ CONDITIONS

15) C++ SWITCH

16) C++ WHILE LOOP

17) C++ FOR Loop

18) C++ BREAK/CONTINUE

19) C++ ARRAYS

20) C++ STRUCTURES

21) C++ REFERENCES

22) C++ POINTERS

23) C++ FUNCTIONS

24) C++ FUNCTION PARAMETERS

25) C++ FUNCTION OVERLOADING

26) C++ RECURSION

Page No.
Date

C++ CLASSES

27) C++ OOP

28) C++ CLASSES / OBJECTS

29) C++ CLASS METHODS

30) C++ CONSTRUCTORS

31) C++ ACCESS SPECIFIERS

32) C++ ENCAPSULATION

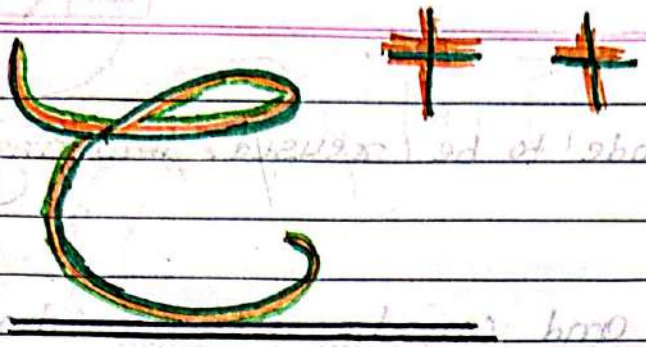
33) C++ INHERITANCE

34) C++ POLYMORPHISM

35) C++ FILES

36) C++ EXCEPTIONS

Page No.
Date



C++ Introduction:

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.

Why Use C++

C++ is one of the world's most popular programming language.

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

Page No.
Date

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.

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.

C++ Get Started :

To start using C++, you need main two things :

- A text editor, like Notepad, to write C++ Code.
- A compiler, like GCC, to translate the C++ code into a language that the computer will understand.

Page No.
Date

C++ Syntax :

Example ↴

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

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

C++ Output :

Example ↴

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

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

practice Example ↴

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

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

C++ Comments :

Comments can be used to explain C++ code, and to make it more readable. It can also be used to prevent execution when testing alternative code. Comments can be Singled-lined or Multi-lined.

Single-line Comments :

Single-line comments start with two forward slashes (//).

Any text between // and the end of the line is ignored by the compiler.

This example uses a single-line comment before a line of code:

Example ↴

```
// This is a comment
cout << "Hello World!";
```

Page No.
Date

Multi-line Comments :

Multi-line comments start with /* and ends with */.

Any text between /* and */ will be ignored by the compiler:

Example :

```
/* The code below will print the words
Hello World!
to the screen, and it is amazing
*/
cout << "Hello World!";
```

C++ Variables :

Variables are containers for storing data values.

In C++, there are different types of variables for example:

- int → stores integers (whole numbers), without decimals, such as 123 or -123
- double → stores floating point numbers, with decimals, such as 19.99 or -19.99

Page No.
Date

- char → Stores single characters, such as 'a' or 'B'. char values are surrounded by single quotes.
- String → Stores text, such as "Hello World". String values are surrounded by double quotes.
- bool → Stores values with two states: true or false.

Declaring variables:

To create a variables, specify the type and assign it a value:

Syntax ↴

```
type variableName = value;
```

Example ↴

Create a variable called myNum of type int and assign it the value 15:

```
int myNum = 15;
cout << myNum;
```

Page No.
Date

Other Types

Example ↓

```
int myNum = 5;                    // Integer (whole number without decimals)
double myFloatNum = 5.99;         // Floating point number (with decimals)
char myLetter = 'D';              // character
string myText = "Hello";          // String (text)
bool myBoolean = true;            // Boolean (true or false)
```

Display Variables :

The cout object is used together with the << operator to display variables.

To combine both text and a variable, separate them with the << operator:

Example ↓

```
int myAge = 35;
cout << "I am " << myAge << " years old.";
```

Page No.
Date

Add variables Together :

Example ↓

```
int x = 5;
int y = 6;
int sum = x + y;
cout << sum;
```

C++ Declare Multiple variables :

To declare More than one variable of the same type, use a comma-separated list:

Example ↓

```
int x = 5, y = 6, z = 50;
cout << x + y + z;
```

One Value to Multiple variables :

Assign the same value to multiple variables in one line:

Example ↓

```
int x, y, z;
x = y = z = 50;
cout << x + y + z;
```

Page No.
Date

C++ Identifiers :

All C++ variables must be identified with unique names.

These unique names are called IDENTIFIERS.

Identifiers can be short names (like x and y) and more descriptive names (age, sum, totalvolume).

Example ↓

```
// Good
int minutesPerHour = 60;

// OK, but not so easy to understand what m actually is
int m = 60;
```

C++ Constants :

When you do not want others to change existing variable values, use the const keyword (this will declare the variable as "constant", which means unchangeable and read-only):

Example ↓

```
const int myNum = 15; // myNum
```

Will always be 15
myNum = 10; // error : assignment of read-only variable 'myNum'

C++ User Input :

Cin is a predefined variable that reads data from the keyboard with the extraction operator (>>)

Example 1

```
int x;
cout << "Type a number: "; //
Type a number and press enter
cin >> x; // Get user input from
the keyboard
cout << "Your number is: " << x;
// Display the input value
```

★ Good To Know

Cout is pronounced "See-out". Used for output, and uses the insertion operator (<<)

Cin is pronounced "See-in". Used for input, and uses the extraction operator (>>)

Page No.
Date

Example ↓

```
int x, y;
int sum;
cout << "Type a number :" ;
cin >> x;
cout << "Type another number : " ;
cin >> y;
sum = x + y;
cout << "Sum is : " << sum;
```

C++ Data Types :

A variable in C++ must be a specified data type:

Example ↓

```
int myNum = 5;
Integer ( Whole number)
float myFloatNum = 5.99 ;
Floating point number
double myDoubleNum = 9.98 ;
Floating point number
char myLetter = 'D' ;
character
bool myBoolean = true ;
Boolean
```

Page No.
Date

String myText = "Hello"; // String

Basic Data Types :

Data Types	size	Description
boolean	1 byte	Stores true or false values
char	1 byte	Stores a single character/letter/number, or ASCII values
int	2 or 4 bytes	Stores whole numbers, without decimals
float	4 bytes	Stores fractional numbers, containing one or more decimals. Sufficient for storing 6-7 decimal digits.
double	8 bytes	Stores fractional numbers, containing one or more decimals. Sufficient for storing

Page No.
Date

Scientific Numbers :

A Floating point numbers can also be a scientific number with an "e" to indicate the power of 10:

Example ↴

```
float f1 = 35e3;
double d1 = 12E4;
cout << f1;
cout << d1;
```

C++ Boolean Data Types :

A boolean data type is declared with the bool keyword and can only take the values true or false.

When the value is returned true = 1 and false = 0.

Example ↴

```
bool isCodingFun = true;
bool isFishTasty = false;
cout << isCodingFun; // outputs
1 (true)
```

Page No.
Date

```
cout << isFishTasty ; // outputs
0 (false)
```

C++ Character Data Types :

The char data types is used to store a single character. The character must be surrounded by single quotes, like 'A' or 'c'.

Example ↓

```
char myGrade = 'B' ;
cout << myGrade;
```

C++ Operators :

Operators are used to perform operations on variables and values.

We use the + operator to add together two values:

Example ↓

```
int x = 100 + 50;
```

Page No.
Date

ex-②

```
int sum1 = 100 + 50;          //
150 (100 + 50)
int sum2 = sum1 + 250;        //
400 (150 + 250)
int sum3 = sum2 + sum2;       //
800 (400 + 400)
```

Arithmetic Operators :

Arithmetic operators are used to perform common Mathematical operations.

Operator	NAME	Description	Example
+	Addition	Adds together two values	x+y
-	Subtraction	Subtracts one value from Another	x-y
*	Multiplication	Multiplies two values	x*y
/	Division	Devides one	x/y

Page No.
Date

		value by Another	
%%	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

Assignment Operators :

Assignment Operators are used to assign values to variables.

In the example below, we use the assignment operator (=) to assign the value 10 to a variable called x :

Example 1,

```
int x = 10;
```

Page No.
Date

List of all assignment Operators :

Operator	Example	Same As
=	x = 5	x = 5
+=	x += 3	x = x + 3
-=	x -= 3	x = x - 3
*=	x *= 3	x = x * 3
/=	x /= 3	x = x / 3
%=	x %= 3	x = x % 3
&=	x &= 3	x = x & 3
|=	x |= 3	x = x | 3
^=	x ^= 3	x = x ^ 3
>>=	x >>= 3	x = x >> 3
<<=	x <<= 3	x = x << 3

Page No.
Date

Comparison Operators :

Comparison Operators are used to compare two values (or variables). This a important in programming.

Example ↓

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

List of all comparison operators :

Operator	Name	Example
==	Equal to	x == y
!=	Not equal	x != y
>	Greater than	x > y
<	Less than	x < y
>=	Greater than or equal to	x >= y
<=	Less than or equal to	x <= y

Page No.
Date

Logical Operators :

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

Operator	Name	Description	Example
&&	Logical and	Returns true if both statements are true	x<5 && x<10
||	Logical or	Returns true if one of the statements is true	x<5 || x<4
!	Logical Not	Reverse the result, returns false if the result is true	!(x<5 && x<10)

STRINGS :

Strings are used to storing text.

A String variable contains a collection of

Characters surrounded by double quotes:

Example ↓

Create a variable of type String and assign it a value.

```
String greeting = "Hello";
```

To use strings, you must include an additional header file in the source code, the <String> libarary.

```
// Include the string library
# include <string>

// create a string variable
String greeting = "Hello";
```

String Concatenation:

The + Operator can be used between strings to add them together to make new String. This is called Concatenation.

Example ↓

```
String firstName = "John";
```

Page No.
Date

```
String LastName = "Doe";
String fullName = firstName +
lastName ;
cout << fullName;
```

Numbers and Strings :

Adding Numbers and Strings

C++ uses the + operator for both addition and concatenation.

Numbers are added. Strings are concatenated. If you add two numbers, the result will be a number.

Example ↴

```
int x = 10;
int y = 20;
int z = x+y ;   // z will be
30 (an integer)
```

If you add two strings, the result will be a string concatenation.

Example ↴

```
String x = "Lo";
```

```
String y = "20";
String z = x + y ; // z will be
1020 (a String)
```

String Lenght :

To get the lenght of a String, use the (lenght() Function.

Example 1,

```
String txt =
"ABCDEFGHIJKLMNOPQRSTUVWXYZ";
cout << " The lenght of the txt
String is : " << txt.lenghth();
```

Access Strings :

The character in a Strings by refferring to its index number inside square brackets [

Example 1,

```
String myString = "Hello";
cout << myString[0];
// Outputs H
```

Change String characters :

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

Example ↓

```
String mystring = "Hello";
mystring[0] = 'J';
cout << mystring;
// outputs Jello instead of Hello
```

Special characters :

Strings - Special characters :

C++ will misunderstand this : String txt, and generate an error

```
String txt = "We are the so-
called "vikings" from the
north.";
```

Backslash escape character (\)

The backslash (\) escape character turns special characters into string characters.

Page No.
Date

Escape character	Result	Description
\'	'	Single quote
\"	"	Double quote
\\	\	Backslash

Example :-

```
String txt = "We are the so-
called \"Vikings\" from the
north.";
```

User Input Strings :

It is possible to use the extraction operator >> on cin to store a string entered by a user.

Example :-

```
string firstName;
cout << "Type your first name :";
cin >> firstName; // get user input
from the keyboard
cout << "your name is : " <<
firstName;
```

Page No.
Date

// Type your first name : John
// your name is : John

String Namespace :

Omitting Namespace :

The using namespace std line can be omitted and replaced with the std line can be keyword, followed by the :: operator for string (and cout) objects.

Example

```
#include <iostream>
# include <String>
int main ()
{
Std :: String greeting = "Hello";
Std :: cout << greeting;
return 0;
}
```

Page No.
Date

C++ MATH :

Max and min

The max(x,y) function can be used to find the highest value of x and y.

Example :

```
cout << max(5,10);
```

C++ <cmath> Header

Other functions, such as sqrt (square root), round (rounds a number) and log (natural) (logarithm), can be found in the <cmath> header file:

Example :

```
// Include the cmath library
# include <cmath>

cout << sqrt(64);
cout << round(2.6);
cout << log(2);
```

Page No.
Date

C++ Booleans

In programming, you will need a data type that can only have one of two values, like:

- Yes / No
- ON / OFF
- True / False

For this, C++ has a bool data type, which can take the values true (1) or False (0)

Boolean Values

A boolean varible is declared with the bool keyword and can only take the values true or False.

Example ↓

```
bool isCodingFun = true;
bool isFishTasty = False;
Cout << isCodingFun; // outputs
1 (true)
Cout << isFishTasty; // outputs
0 (false)
```

From the example above, you can read that a true value returns 1, and False returns 0.

However, it is more common in to return a boolean value by comparing values and variables

C++ Boolean Expressions :

A Boolean expression returns a boolean value that is either 1 (true) or 0 (False).

This is useful to build logic, and find answers.

Use comparison operator, such as the greater than (>) operator, to find out if an expression (or variable) is true or false.

Example :

```
int x = 10;
int y = 9;
cout << (x > y); // returns 1
(true), because 10 is higher than
9
```

Real Life Example

"Real life example" where we need to find out if a person is old enough to vote.

In the example below, we use the >= comparison operator to find out if the age (25)

is greater than OR equal to the voting age limit, which is set to 18.

Example ↓

```
int myage = 25;
int votingAge = 18;

cout << (myAge >= votingAge); //
returns 1 (true), meaning 25 year
olds are allowed to vote!
```

C++ If... Else:

C++ Conditions and If Statements

C++ supports the usual logical conditions from Mathematics:

- Less than: a < b
- Less than or equal to: a <= b
- Greater than: a > b
- Greater than or equal to: a >= b
- Equal to: a == b
- Not Equal to: a != b

C++ has following conditional statements:

- Use if to specify a block of code to be

executed, if a specified condition is true.

- Use else to specify a block of code to be executed, if the same condition is False
- Use else if to specify a new condition to test, if the first condition is false
- Use switch to specify many alternative blocks of code to be executed.

The if Statement

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

Syntax

```
if (condition)
{
// block of code to be executed
if the condition is true
}
```

* IMPORTANT → if is in lowercase letters. Uppercase letters (IF or If) will generate an error.

Page No.
Date

Test two values to find out if 20 greater than 18. If the condition is true, print some text:

Example ↓

```
if (20 > 18)
{
cout << "20 is greater than 18";
}
```

C++ Else :

The else statement

Else statement to specify a block of code to be executed if the condition is False.

Syntax

```
if (condition)
{
 // block of code to be executed if the condition is true
} else {
 // block of code to be executed if the condition is false.
}
```

Example ↓

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

C++ Else If :

The else if statement

Use the else if statement to specify a new condition if the first condition is false.

Syntax

```
if (condition) {
  // 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 the condition1 is false and
```

condition2 is False
}

Example ↓

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

C++ Short Hand If Else :

Short Hand If ... Else (Ternary Operator)

There is also a shor-hand if else, which is known as the ternary operator. because it 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.

Syntax

```
variable = (condition) ?
expressionTrue : expressionFalse;
```

Page No.
Date

Example 7

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

C++ SWITCH :

C++ Switch Statements

switch statement to select one of many code to be executed.

Syntax

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

This is how it Works :

- The switch expression 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 this chapter.

Example 7

```
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;
```

Page No.
Date

```
    cout << "Friday";
    break;
}
// outputs "Thursday" (day 4)
```

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.

The default Keyword

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

Example ↓

```
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 ";
}
// outputs " Looking forward to
the Weekend "
```

C++ While Loop:

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.

The While loop loops through a block of code as long as specified condition is true.

Syntax

```
While (condition) {
   // code block to be executed
}
```

Example ↴

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

C++ Do/While Loop :

The Do/While Loop

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.

Syntax

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

Example ↴

```
int i = 0;
do {
```

```
cout << i << "\n";
  i++;
}
while (i<5);
```

C++ For Loop :

Use the for loop instead of a while loop.

Syntax

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

Example 1

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

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".

Example ↓

```
// 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)
  }
}
```

The foreach Loop

There is also a "for-each loop" which is used exclusively to loop through elements in an array.

Syntax

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

Example ↓

```
int myNumbers [5] = {10, 20, 30, 40, 50, };
```

Page No.
Date

```
for(int i : myNumbers) {
  cout << i << "\n";
}
```

C++ Break and Continue:

Break Statement used in an earlier chapter of this tutorial. It was used to "jump out" of a Switch Statement.

The break statement can also be used to jump out of a loop.

Example ↓

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

C++ Continue

The continue Statements breaks one iteration if a specified condition occurs, and continues with the next iteration in the loop.

Page No.
Date

Example ↓

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

Break and Continue in While Loop

Use break and continue in while loops.

Break Example ↓

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

Continue example ↓

```
int i = 0;
while (i < 10) {
  if (i == 4) {
```

Page No.
Date

```
    i++;
    continue;
  }
  cout << i << "\n";
  i++;
}
```

C++ Arrays

Arrays are used to store multiple values in a single variable, instead of declaring separate 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];
```

Access the Elements of an Array

Access an array element by referring to the index number inside square brackets [].

This statement accessess the value of the first element in cars:

Example ↓

```
String cars[4] = {"Volvo", "BMW",
"Ford", "Mazda"};
cout << cars[0];
// outputs Volvo
```

Change an Array Element

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

```
cars[0] = "Opel";
```

Example ↓

```
String cars[4] = {"Volvo", "BMW",
"Ford", "Mazda"};
cars[0] = "Opel";
cout << cars[0];
// Now outputs Opel instead of
Volvo.
```

Loop Through an Array

Loop through the array elements with the For loop.

Page No.
Date

Example ↓

```
String cars[5] = {"Volvo", "BMW",
"Ford", "Mazda", "Tesla"};
for (int i = 0; i < 5; i++) {
  cout << cars[i] << "\n";
}
```

C++ Omit Array Size :

Omit Array Size

The compiler is smart enough to determine the size of the array based on the number of inserted values:

```
String cars[] = {"Volvo", "BMW",
"Ford"}; // Three array elements
```

Example ↓

```
String cars[3] = {"Volvo", "BMW",
    ; // Also three array
elements.
```

Page No.
Date

C++ Array Size

Get the Size of an Array

To get the size of an array, you can use the sizeof() operator:

Example ↓

```
int myNumbers[5] = {10, 20, 30,
40, 50, };
cout << sizeof(myNumbers);
```

Result:
20

Loop Through an Array With sizeof()

In Arrays and Loops chapter, we wrote the size of the array in the loop condition (i < 5). This is not ideal, since it will only work for arrays a specified size.

By using the sizeof() approach from the example above, we can now make loops that work for arrays of any size, which is more sustainable.

Example ↓

```
int myNumbers[5] = { 10, 20, 30,
```

Page No.
Date

```
40, 50};
for (int i = 0; i <
sizeof(myNumbers) / sizeof(int);
i++) {
  cout << myNumbers[i] << "\n";
}
```

C++ Multi-Dimensional Arrays:

Multi-Dimensional Arrays

A multi-dimensional array is an array of arrays.

To declare a multi-dimentional array, define the variable type, specify the name of the array followed by square bracket which specify how many elements the main arrays has, followed by another set of square brackets which indicates how many elements the sub-arrays have.

```
string letters[2][4];
string letters[2][4] = {
 {"A", "B", "C", "D"},
 {"F", "F", "G", "H"}
};
```

Arrays can have any number of dimensions. The more dimentions an array has, the more complex the code becomes.

```
String letters [2] [2] [2] = {
  {
    { "A" , "B" },
    { "C" , "D" }
  },
  {
    { "E" , "F" },
    { "G" , "H" }
  }
};
```

Access the Elements of a Multi-dimensional Array

To access an element of a multi-dimensional array, specify an index number in each of the array's dimensions.

This statement accesses the value of the element in the first row (0) and third column (2) of the letters array.

Example ↓

```
String letters [2] [4] = {
  { "A", "B", "C", "D" },
  { "E", "F", "G", "H" }
};
cout << letters [0] [2] ; // outputs "C"
```

Page No.
Date

Change Elements in a Multi-Dimensional Array

To change the value of an element, refer to the index number of the element in each of the dimensions.

Example ↓

```
String letters [2][4] = {
{"A", "B", "C", "D",},
{"E", "F", "G", "H",}
};
letters [0] [0] = "Z";
cout << letters [0] [0]; // Now outputs "Z" instead of "A"
```

Loop Through a Multi-dimensional Array

To loop through a multi-dimentional array, you need one loop for of the array's dimensions.

Example ↓

```
Sting letters [2] [4] = {
  {"A", "B", "C", "D"},
  {"E", "F", "G", "H"}
};
for (int i = 0; i<2; i++) {
```

Page No.
Date

```
for (int j = 0; j < 4; j++) {
cout << letters [i][j] <<
"\n";
    }
}
```

Why Multi-Dimensional Arrays?

Multi dimensional arrays are great at representing grids. This example shows a practical use for them. Multi-dimensional array to represent a small game of Battleship.

C++ Structures (Struct)

C++ Structures

Structures are a way to group several related variables into one place. Each variable in the structure is known as a Member of the structure.

Unlike an array, a structure can contain many different data types (int, string, bool, etc.)

Create a Structure

To create a structure, use the struct keyword and declare each of its members inside curly braces.

Page No.
Date

The declaration, specify the name of the structure variable (myStructure in the below example ↓

```
Struct {                    // Structure declaration
int myNum;                  // Member (int variable)
String myString;            // Member (String variable)
} myStructure;              // Structure variable
```

Access Structure Members

To access members of a structure, use the dot syntax (.);

Example ↓

```
// Create a structure variable called myStructure
Struct {
  int myNum;
  String myString;
} myStructure;

// Assign values to members of myStructure
myStructure.myNum = 1;
myStructure.myString = "Hello World!";
```

Page No.
Date

```
// print members of myStructure
cout << myStructure.myNum <<
"\n";
cout << myStructure.myString <<
"\n";
```

One Structure in Multiple Variables

Use a comma (,) to use one structure in many variables.

```
Struct {
    int myNum
    String myString;
} myStruct1, myStruct2,
myStruct3; // Multiple Structure
variable separated with commas
```

Named Structure

By giving a name to the structure, you can treat it as a data type. This means that you can create variables with this structure anywhere in the program at any time.

To create a named structure, put the name of the structure right after the struct keyword.

Page No.
Date

```
struct myDataType { // This
structure is named "myDataType"
int myNum;
    string myString;
};
```

C++ References:

Creating References

A reference variable is a "reference" to an existing variable, and it is created with the & operator.

```
string Food = "pizza"; // Food
variable
string &meal = Food; // //
reference to food
```

Example ↓

```
string Food = "pizza";
string &meal = Food;
cout << food << "\n"; // Outputs
pizza
cout << meal << "\n"; // Outputs
pizza
```

Page No.
Date

C++ Memory Address :

Memory Address

When a variable is created in c++, a memory address is assigned to the variable. And when we assign a value to the variable, it is stored in this memory address.

Use the & operator, and the result will represent where the variable is stored.

Example↓

```
String Food = "pizza";
Cout << &food; // outputs
0x6dfed4
```

C++ pointers :

Creating pointers

Memory address of a variable by using the & operator:

Example↓

```
String food = "pizza"; // A food
variable of type string
```

Page No.
Date

cout << food; // outputs the value of food (pizza)
cout << &food; // outputs the memory address of food (0x6dfed4)

C++ Deference

Get Memory Address and Value

Use the pointer to get the value of the variable, by using the * operator.

Example-

```
string food = "pizza"; // variable declaration
string* ptr = &food; // pointer declaration

// Reference: output the memory address of food with the pointer (0x6dfed4)
cout << ptr << "\n";

// Deference: output the value of food with the pointer (pizza)
cout << *ptr << "\n";
```

Page No.
Date

C++ Functions :

A function is a block of code which only runs when it is called.

Functions are used to perform certain actions, and they are important for reusing code.

Create a Function

C++ provides some pre-defined functions, such as main(), which is used to execute code. But you can also create own functions to perform certain actions.

Syntax

```
void myfunction () {
  // code to be executed.
}
```

Example ↓

```
// create a function
void myfunction() {
  cout << "I just got executed!";
}

int main () {
  myfunction(); // call the
function
```

Page No.
Date

```
    return 0;
}
// outputs "I just got executed!"
```

Function Declaration and Defination

- Declaration - The return type, the name of the function, and parameters
- Defination - The body of the function (code to be executed)

```
void myFunction() { // declaration
  // the body of the function (defination)
}
```

Example ↓

```
int main() {
  myFunction();
  return 0;
}

void myFunction() {
  cout << "I Just got executed!";
}

// Error
```

Page No.
Date

C++ Function parameters :

Parameters and Arguments

Parameters act as variables inside the function.

Parameters are specified after the function name, inside the parentheses.

Syntax

```
void functionName (parameter1, parameter2, parameter3) {
  // Code to be executed
}
```

Example ↓

```
void myFunction (String fname) {
    cout << fname <<
}

    main () {
  myFunction ("Liam");
  myFunction ("Jenny");
  myFunction ("Ania");
   return 0;
}

 // Liam Refsnes
 // Jenny Refsnes
 // Ania Refsnes
```

Page No.
Date

C++ Default parameters

Default parameter Value

Default parameter value also use, by using the equals sign (=)

It uses also default value ("Norway");

Example 7

```
void myFunction(String country =
"Norway") {
  cout << country << "\n";
}

int main() {
  myFunction("Sweden");
  myFunction("India");
  myFunction();
  myFunction("USA");
  return 0;
}

// Sweden
// India
// Norway
// USA
```

Page No.
Date

C++ Multiple Parameters :

Multiple parameters

In this parameter you can add as many parameters as you want.

Example :

```
void myFunction (string Fname, int
age) {
  cout << Fname << " Refsnes. "
<< age << " years old. \n";
}

  main () {
  myFunction ("Liam", 3);
  myFunction ("Jenny", 14);
  myFunction ("Ania", 30);
    return 0;
}
```

```
// Liam Refsnes. 3 years old.
// Jenny Refsnes. 14 years old.
// Ania Refsnes. 30 years old.
```

C++ Return Keyword :

Return Values

The void keyword, used in the previous page,

Page No.
Date

Indicates that the function should not return a value. instead of void, and use the return keyword inside the function.

Example ↓

```
int myFunction( int x) {
    return 5+x;
}

int main() {
  cout << myFunction(3);
    return 0;
}

// Outputs 8 (5+3)
```

C++ Functions - Pass By Reference

Pass By Reference

This can be useful when you need to change the value of the arguments.

example ↓

```
void swapNums (int &x, int &y) {
  int z = x;
  x = y;
  y = z;
}
```

```
int main () {
    int firstNum = 10;
    int secondNum = 20;

    cout <<"Before swap: " <<
"\n";
    cout << firstNum << secondNum
<<"\n";

    // Call the function, which
will change the values of
firstNum and secondNum
    swapNums (firstNum, secondNum);

    cout << "After swap: " <<"\n";
    cout << firstNum << secondNum
<<"\n";

    return 0;
}
```

C++ pass Array to a function:

pass Arrays as Function parameters

Example ↓

```
void myFunction ( int myNumbers [ ])
{
```

Page No.
Date

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

int main () {
int myNumbers [5] = { 10, 20, 30,
40, 50 };
myFunction (myNumbers);
return 0;
}
```

C++ Function Overloading :

Function Overloading :

Function overloading, multiple functions can have the same name with different parameters:

Example :

```
int myFunction ( int x)
Float myFunction ( float x)
double myFunction ( double x,
double y)
```

C++ Recursion

Recursion is the technique of making a function call itself.

This technique provides a way to break complicated problems down into simple problems which are easier to solve.

Example ↴

```
int sum( int k){
  if (k>0) {
    return k + sum(k-1);
  } else {
    return 0;
  }
}

int main () {
  int result = sum (10);
  cout << result;
  return 0;
}
```

C++ OOP

C++ What is OOP ?

OOP stands for object - oriented programming.

Procedural programming is about writing procedures or functions that perform operations on the data, while object - oriented programming is about

Page No.
Date

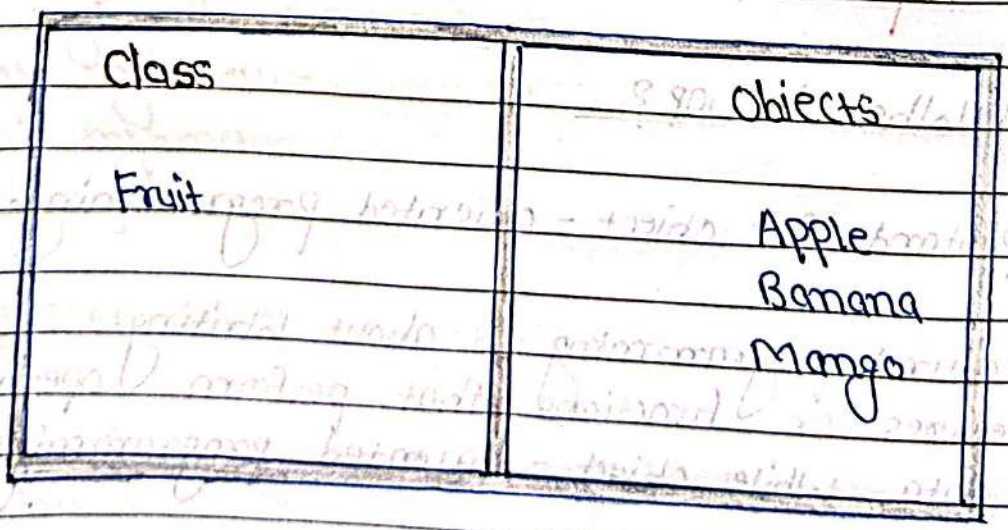
creating objects that contain both data and functions.

Advantages over Procedural programming :

- OOp is Faster and easier to execute
- OOP provides a clear structures for the programs.
- OOP helps to keep the C++ code DRY "Dont' Repeat yourself", and makes the code easier to maintain, modify and debug
- OOP makes it possible to create full reusable applications with less code and shorter development time.

C++ What are classes and objects?

Classes and objects are the two main aspects of object - oriented programming.



Page No.
Date

C++ Classes and Objects :

C++ Classes / Objects

C++ is an object-oriented programming language.

Everything in C++ is associated with classes and objects, along with its attributes and methods. For example → in real life, a car is an object. The car has attributes, such as weight and color, and methods, such as drive and brake.

Create a Class

Use the class keyword.

Example :

```
Class Myclass {  // The class
    public:      // Access specifier
        int myNum;  // Attribute (int variable)
        String myString;  // Attribute (String variable)
};
```

Page No.
Date

Create an Object :

In C++, an object is created from a class. We have already created the class named Myclass, so now we can use this to create objects.

To create an object of Myclass, specify the class name, followed by the object name.

To access the class attributes (myNum and myString), use the dot syntax (.) on the object:

Example ↴

```
class Myclass {     // The
class
  public:           // Access
Specifier
    int myNum;
Attribute (int variable)
    string myString; //
Attribute (String variable)
};

int main() {
  Myclass myObj;  // Create an
Object of Myclass.

  // Access attributes and set values
```

```
myobi.myNum = 15;
myobi.myString = "Some text";

// print attribute values
cout << myobi.myNum << "\n";
cout << myobi.myString;
return 0;
}
```

Multiple Objects

```
// Create a car class with some attributes
class Car {
  public
  String brand;
  String model;
  int year;
};

int main() {
// Create an object of Car
Car carObj1;
carObj1.brand = "BMW";
carObj1.model = "X5";
carObj1.year = 1999;

// Create another object of Car
Car carObj2;
```

```
carobi2.brand = "Ford";
carobi2.model = "Mustang";
carobi2.year = 1969;

// print attribute values
cout << carobi1.brand << " " <<
carobi1.model << " " <<
carobi1.year << "\n";
    cout << carobi2.brand << " " <<
carobi2.model << " " <<
carobi2.year << "\n";
    return 0;
}
```

C++ Class Methods :

Class Methods

Methods are Functions that belongs to the class.

There are two ways to define functions that belongs to a class.

- Inside class defination
- Outside class defination

We define a function inside the class, and name it "myMethod".

Page No.
Date

Inside Example ↓

```
class Myclass {        // The class
  public:              // Access specifier
    void myMethod () { // Method / function defined inside the class
      cout << "Hello World!";
    }
};

int main () {
  Myclass myobi;       // Create an object of Myclass
  myobi.myMethod ();   // Call the method
  return 0;
}
```

Outside Example ↓

```
class Myclass {        // The class
  public:              // Access specifier
    void myMethod ();  // Method / function declaration
};
```

Page No.
Date

```
// Method / Function difinition outside the class
void myclass::myMethod () {
  cout << "Hello World";
}

int main () {
  Myclass myobi; // create an object of myclass
  myobi.myMethod (); // call the method
  return 0;
}
```

Parameters

Example ↓

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

class car {
  public:
    int speed ( int maxspeed );
};
int car::speed ( int maxspeed );
  return maxspeed;
}
int main () {
```

Page No.
Date

```
Car myObj; // Create an object
of Car
  Cout << myObj.speed(200); //
call the method with an argument
  return 0;
}
```

C++ Constructors :

Constructors

A constructor in C++ is a special method that is automatically called when an object of a class is created.

To create a constructor, use the same name as the class, followed by parentheses ():

Example :

```
Class MyClass {      // The class
  public:            // Access
specifier
  MyClass() {        // Constructor
    Cout << "Hello World!";
  }
};
int main() {
```

Page No.
Date

```
    MyClass myobj; // create an
object of MyClass (this will call
the constructor)
    return 0;
}
```

Constructor parameters

Constructors can also take parameters (just like regular functions), which can be useful for setting initial values for attributes.

brand, model and year attributes, and a constructor with different parameters. Inside the constructor we set the attributes equal to the constructor parameters (brand = x,)

Example 7

```
class car {          // The class
  public:            // Access
Specifier
    String brand;    // Attribute
    String model;    // Attribute
    int year;        // Attribute
    car (String x, String y, int
z) { // Constructor with
parameters
      brand = x;
```

```
        model = y;
        year = z;
    }
};

int main () {
  // create car objects and call the
constructor with different values

car Carobj1 ("BMW", "X5", 1999);
car Carobj2 ("Ford", "Mustang",
1969);

// print values
  cout << Carobj1.brand << " " <<
Carobj1.model << " " <<
Carobj1.year << "\n";
  cout << Carobj2.brand << " " <<
Carobj2.model << " " <<
Carobj2.year << "\n";
  return 0;
}
```

C++ Access Specifiers :

Access Specifiers

You are quite familiar with the public keyword that appears in all of our class examples.

Page No.
Date

C++ Inheritance :

Inheritance

In C++, it is possible to inherit attributes and methods from one class to another. We group the "inheritance concept" into two categories.

- derived class (child) - the class that inherits from another class
- base class (parent) - the class being inherited from

To inherit from a class, use the : symbol

In the example below, the car class (child) inherits the attributes and methods from the vehicle class (parent):

Example ↓

```
// Base class
class vehicle {
public:
  String brand = "Ford";
  void honk() {
    cout << " Tuut, tuut! \n";
  }
};
```

```
private:
    // private attribute
    int salary;

public:
    // setter
    void setsalary (    s) {
        salary = s;
    }
};
    int main () {
    Employee myobi;
    myobi. setsalary (50000);
    cout << myobi. getsalary ();
        return 0;
}
```

Why Encapsulation ?

- It is considered good practice to declare your class attributes as private. Encapsulation ensures better control of your data, because you can change one part of the code without affecting other parts.

- Increased security of data.

Page No.
Date

Example ↴

```
class Myclass {      // The class
  Public:            // Access
Specifier
    // class members goes here
};
```

C++ Encapsulation :

Encapsulation

The meaning of Encapsulation, is to make sure that "sensitive" data is hidden from users. To achieve this, you must declare class variables / attributes as private. you can provide public get and set methods.

Access Private Members

To access a private attribute, use Public "get" and "set" methods.

Example ↴

```
#include <iostream>
using namespace std;
class Employee {
```

```
void myFunction () {
  cout << " some content in
  parent class ;
  }
};

// Another base class
class myotherclass {
  public:
    void myOtherFunction () {
      cout << " Some content in
another class " ;
    }
};
  // Derived class
class Mychildclass : public
Myclass, public myotherclass {
};

int main () {
  Mychildclass myobi;
  myobi.myFunction ();
  myobi. myotherfimction ();
    return 0;
}
```

C++ Inheritance Access

Access Specifiers

Page No.
Date

```
        cout << "some content in
parent class.";
    }
};
// Derived class (child)
class Mychild: public Myclass {
};

// Derived class (grandchild)
class MyGrandchild : public
Mychild {
};

int main () {
  MyGrandchild myobi;
  myobi. myfunction ();
  return 0;
}
```

C++ Multiple Inheritance :

Mutiple Inheritance

Using a comma - separated list.

Example :

```
// Base class
class MyClass {
 public :
```

```
// Derived class
class car: public vehicle {
public:
    String model = "Mustang";
};

int main() {
    Car mycar;
    mycar.honk();
    cout << mycar.brand + "brand" +
mycar.model;
    return 0;
}
```

C++ Multilevel Inheritance

Multilevel Inheritance

A class can also be derived from one class, which is already derived from another class.

In the following example, MyGrandchild is derived from class MyChild

Example:

```
// Base class (parent)
class Myclass {
  public:
    void myFunction () {
```

Page No.
Date

```
class pig : public Animal {
  public:
  void animalSound() {
    cout << "The pig says: wee
wee\n";
  }
};

// Derived class
class Dog : public Animal {
public:
  void animalSound() {
    cout << "The dog says: bow
wow\n";
  }
};
```

C++ Files :

C++ Files

The fstream libary allows us to work with files.

To use the fstream library, include both the standard <iostrem> AND the <fstream> header file.

Page No.
Date

```
    myObj.bonus << "\n";
    return 0;
}
```

C++ Polymorphism :

Polymorphism

Polymorphism means "many forms", and it occurs When we have many classes that are related to each other by inheritance.

For example, think of a base class called Animal that has a method called animalSound(). Derived classes of Animal could be Pigs, Cats, Dogs, Birds - And they also have their own implementation of an animal sound.

Example ↓

```
// Base class
class Animal {
  public:
    void animalSound() {
    cout << "The animal makes a
    sound\n";
    }
};
// Derived class
```

Page No.
Date

Example 1.

```
// Base class
class Employee {
 protected: // protected access
specifier
    int salary;
};

// Derived class
class programmer: public Employee
{

  public:
     int bonus;
    void setsalary (int s){
        Salary = s;
    }
    int getsalary () {
      return Salary;
    }
};

int main () {
  programmer myobi;
  myobi. setSalary (50000 );
  myobi. bonus = 15000 ;
  cout << "Salary:" <<
myobi. getsalary () << "\n";
    cout << "Bonus:" <<
```

Example ↓

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

There are three classes included in the fstream library, which are used to create, write or read files.

Class	Description
ofstream	Creates and writes to files
ifstream	Read from files
fstream	A combination of ofstream and ifstream: creates, reads, and writes to files.

Create and Write To a File

To create a file, use either the ofstream or fstream class and, specify the name of the file.

To write to the file, use the insertion operator (<<).

Page No.
Date

Example ↴

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

int main(){
  // create and open a text file
  ofstream
MyFile("filename.txt");

  // Write to the file
  MyFile << "Files can be tricky,
but it is fun enough!";

  // close the File
  MyFile.close();
}
```

Read a File

To read from a file, use either the ifstream or fstream class, and the name of the file.

Note that we also use a while loop together with the getline() function (which belongs to the ifstream class) to read the file line by line, and to print the content of the file.

Page No.
Date

Example 1

```
// Create a text string, which is
used to output the text file
String myText;

// Read from the text file
ifstream
MyReadFile("Filename.txt");

// use a while loop together with
the getline() function to read
the file line by line.
while (getline (MyReadFile,
myText)) {
  // output the text from the
File
  cout << myText;
}

// Close the File
MyReadFile.close();
```

C++ Exceptions

C++ Exceptions

When executing C++ code, different errors can occur: coding errors made by the programmer, errors due to wrong input, or other unforeseeable

```
    throw exception; // Throw an
exception when a problem arise
}

    catch () {
        // Block of code to handle
errors
}
```

Handle Any Type of Exceptions (...)

If you do not know the throw type used in the try block, you can use the "three dots" syntax (...) inside the catch block, which will handle any type of exception.

Example ↓

```
try {
    int age = 15;
    if (age >= 18) {
        cout << "Access granted - you
are old enough.";
    } else {
        throw 505;
    }
}

    catch (...) {
    cout << "Access denied - you must be at least
18 years old.\n";
}
```

Page No.
Date

things.

When an errors occurs, C++ will normally stop and generate an error message. The Technical term for this is: C++ will throw an exception

C++ try and catch

Exception handling in C++ consist of three keywords: try, throw and catch:

The try statement allows you to define a block of code to be tested for errors while it is being executed.

The throw keyword throws an exception when a problem is detected, which lets us create a custom error.

The catch statement allows you to define a block of code to be executed, if an error occurs in the try block.

The try and catch keywords come in pairs.

Example↓

```
try {
  // Block of code to try
```