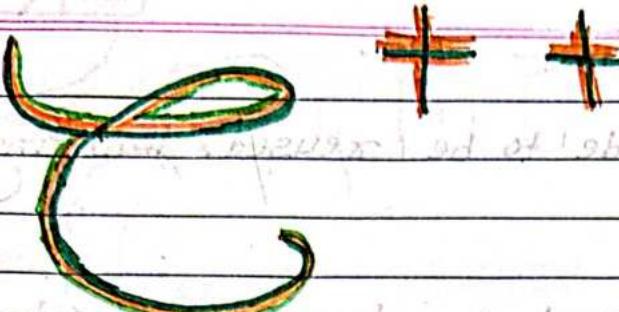


Page No.	
Date	



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.

: bjarne stroustrup

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.

: main function

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++ supports classes and objects, while C does not.

C++ Get Started :

To start using C++, you need mainly 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.			
Data			

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;
```

```
cout << "Hello World!" >> fptr
```

```
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 single-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

- **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 variable, 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;
```

Other Types

Example ↴

```
int mynum = 5;
```

$$11: \bar{x} = x_{\text{mid}}$$

Integer (whole number without decimals) $x + y = \text{sum}$

double myFloatNum = 5.99;

Floating point number (with decimals)

char myLetter = 'Q'; // signifiцира Q + 7
character

```
string myText = "Hello"; // remark at
```

String (Text) incrementation . 9.2.11 start command

```
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 7.

```
int myAge = 35;
```

```
cout << "I am " << myAge << " years old.";
```

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 know what m is actually is
understand what m is 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 : error : assignment
~~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 7

```
int x;
cout << "Type a number: ";
Type a number and press enter and 40.11
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 7

```
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 7

```
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
```

String myText = "Hello"; // 5 bytes

String

Basic Data Types

Data Types : size domain Description part // > fun

boolean 1 Stores true or false

byte 1 Stores values from -128 to 127

char 1 Stores a single character/letter/number,

or ASCII values

int 4 Stores whole numbers, without decimals

bytes

float 4 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

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; // with scientific notation
double d1 = 12E4; // with scientific notation
cout << f1; // will (standard float) represent
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 type 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;
```

Ex-①

int sum1 = 100 + 50;

150 (100 + 50)

int sum2 = sum1 + 250; // result of sum1 + 250

400 (150 + 250)

similarly int sum3 = sum2 + sum1; // result of sum2 + sum1

800 (400 + 400) respectively and so on

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

Divides one

x/y

~~operator ; program firm value by Another~~

%

Modulus

Returns the

x % y

division

Remainder

$$E \% X = X$$

$$E = \% X$$

++

Increment

Increases the

++ x

$$E - X = X$$

$$E = + X$$

--

Decrement

Decreases

= -- x

the Value

$$E = \backslash X$$

of a variable by 1

$$E X = X$$

$$E = \backslash X$$

$$= \backslash$$

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 ↴

int x = 10;

Page No.		
Date		

List of all assignment Operators:

Operator Example Same As

= $x = 5$ $x = 5$

+= $x += 3$ $x = x + 3$

-= $x -= 3$ $x = x - 3$

*= $x *= 3$ $x = x * 3$

/= $x /= 3$ $x = x / 3$

%= $x \% = 3$ $x = x \% 3$

&= $x \& = 3$ $x = x \&$

: Assignment Operator

^= $x ^= 3$ $x = x ^ 3$

>>= $x >> = 3$ $x = x >> 3$

<<= $x << = 3$ $x = x << 3$

Comparison Operators :

Comparison Operators are used to compare two values (or variables). This is 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 $x == y$

\neq Not equal $x \neq y$

$>$ Greater than $x > y$

$<$ Less than $x < y$

\geq Greater than or equal to $x \geq y$

\leq Less than or equal to $x \leq y$

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	$x < 5 \& x < 10$
----	-------------	---------------------------------	-------------------

	Logical or	Returns true if one of the statements	$x < 5 x < 10$
--	------------	---------------------------------------	-------------------

!	Logical Not	Reverse the result,	$!(x < 5 \& x < 10)$
---	-------------	---------------------	----------------------

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> library.

```
// 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 a new string.
This is called Concatenation.

Example ↴

String firstname = "John";

```
String lastname = "Doe";
String fullname = firstname +
lastname;
cout << fullname;
```

#include <iostream> // header file for cout

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"; // " " = empty string operator
```

Page No.	
Date	

String y = "20"; : prints 20 and prints 20

String z = x + y; // z will be 1020

1020 (a string) : prints 1020 and prints 1020

maximum length of string D prints 1020

length of string D prints 1020

String Length :

To get the length of a string, use the `length()` function.

Example ↴

String txt = "ABCD...FGHIJKLMNOPQRSTUVWXYZ";

cout << "The length of the string is: ";

string is: " << txt.length();

Access Strings:

The character in a strings by referring to its index number inside square brackets []

Example ↴

String myString = "Hello";

cout << myString[0];

// outputs H

Page No.		
Date		

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, 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.

Escape character : Result Description

' : ' ' single quote

" : " " double quote

\ : \ backslash Backslash

and more will be explained later after all

Example : with detail knowledge how business

program :: adding \ backslash in front of \\

String txt = "We are the some \\ printed not
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 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;
```

```
}
```

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 off 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.			
Data			

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 variable 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 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; // input taken off screen
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 ↴

: programX7 crashing H.7

```
int myAge = 25;
```

```
int votingAge = 18; // missing method A
```

```
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 : $a \leq b$
- Greater than : $a > b$
- Greater than or equal to : $a \geq b$
- 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 the specified condition is true.

~~test some thing, if it is true, print "It 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

: 32/7 4.2

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.

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

if 20 > 18 then print a message of 20 is great

Example [C++] cout << "20 is greater than 18";

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

{ cout << "20 is greater than 18"; }

cout << "20 is greater than

18";

{ cout << "20 is greater than 18"; }

C++ Else :

frustrated if not

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 7

```
int time = 20;
```

if (time < 18) {

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

```
cout << "Good evening." ; } (a) errit) if
```

// Outputs "Good evening." (e > emit) 76 9219 8

"Upb Load" \Rightarrow fun

C++ Else If : if ".cinious cout" == true

The else if statement

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

Syntax `(#(T) m[0:T])` `m[0:T]` length `4096`

if (condition) {
 code
}

// block of code to be executed

If condition1 is true then print "Hello World"

3 else if (condition2) { }
if (condition1) { }
if (condition2) { }

// block of code to be executed;

If the condition is false and condition is true.

Condition is true
{} else {}

// block of code to be executed

If the condition is false, and $=$ is chosen

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 short-hand ~~if else~~ ^{operator}, 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 ↴

```
int time = 20;
if (time < 18) {
    cout << "Good morning." ;
} 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 for each case.
- 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 ↴

```
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;
}
// outputs "Thursday" (day 4)

```

: yester
: F 9/20
: "yesterday is today" >> two
: apres
: tuesday
: of tomorrow is today" >> three
: "tomorrow after
: 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.

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`

`}`

`cout << " Upcoming 26 robot " >> fin;`

`if (fin == 1)`

`cout << " Upcoming 26 robot " >> fin;`

Example ↴

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

"In" >> i >> fno
i++

i (first) initial

: final val + 1

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; beginning of while loop starts "final val" = 0
do {
    // code block
}
```

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

{

while (i < 5);

C++ For Loop :

Use the for loop instead of a while loop.

Syntax

```
for ( Statement1; Statement2;
```

```
Statement3 ) {
```

```
// 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 ↴

```
{ (for outer loop : i from 1 to 2)
  "Outer" (i is 1)
  {
    // Outer loop
    for (int i=1; i<=2; ++i) {
      cout << "Outer: " << i << "\n";
    }
  }
}
```

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

// Executes 2 times : (iteration 1 and 2nd)

records // Inner loop : i from 1 to 3

```
"if"
for (int j = 1; j<=3; ++j) {
  cout << "Inner: " << j << endl;
}

```

"\n"; // Executes 6 times (2 * 3)

email of 21st May 2010 (inner loop starts after outer loop)

{} (inner loop starts after outer loop)

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

{} (inner loop starts after outer loop)

different from 2nd one without having a

Example ↴ and out of iteration from out

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

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

C++ Break And Continue :

Break Statement used in an earlier chapter of this tutorial. It has been 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.



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) {
```

```

    i++;
    continue;
}
{ (hi > i) && (j == i + 1) } m1
cout << i << "\n";
{ (j == i) } m1
i++;
}
{ (m1 >> i) } m1

```

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 accesses 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.

Example ↴

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

C++ Array size must match with no. of elements

C++ Omit Array size :

Omit Array size must match with no. of elements

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.
```



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:

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 of 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,}
```

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

C++ Multi-Dimensional Arrays:

Multi-Dimensional Arrays

A multi-dimensional array is an array of arrays.

To declare a multi-dimensional array, define the variable type, specify the name of the array followed by square bracket which specifies how many elements the main array 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"},
```

```
{ "F1", "F2", "G", "H"}
```

```
};
```

Addressing scope of different cells in program

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

String letters [2] [2] m [2] = {

```
lett { off. to 2nd row off word of
lett. to { "A", "B" } m[2] off. to address object
{ "C", "D" }
}
{
{ "E", "F" },
{ "G", "H" }
};

{ "H", "P", "Q", "E" }
```

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. In the code "letters[0][2]",

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" = i + 1. m[2]
```

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-dimensional array, you need one loop for each of the array's dimensions.

Example ↴

String letters [2][4] = {

{"A", "B", "C", "D"},

{"E", "F", "G", "H"},

};

for (int i = 0; i < 2; i++) {

Page No.	
Date	

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

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.	
Data	

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` with the help of assignment operator
`myStructure.mynum = 1;` and
`myStructure.myString = "Hello World!";`

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

One Structure in Multiple Variables

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

```
D) Structure
int myNum;
String myString;
{ myStruct1, myStruct2, "prefix" = first print
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.



```
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; // meal is a 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
0xb6fed4

C++ pointers :

Creating pointers

Memory address of a variable by using the & operator:

Example ↴

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

`cout << food; // outputs the value of food (pizza)`

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

C++ Reference

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"; // Reference: Output the memory address of Food with the pointer (0x6dfed4)`

`// Deference: Output the value of food with the pointer (pizza)`

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

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  
}
```

```
    }  
    return 0;
```

: output H2

// outputs "I just got executed!"

Function Declaration and Definition

- Declaration - The return type, the name of the function and parameters

- Definition - The body of the function (Code to be executed)

```
void myFunction() { // part of declaration
```

declaration of function myFunction

// the body of the function
(definition)

```
}
```

Example ↴

```
int main() {
```

myFunction();

return 0;

```
}
```

```
void myFunction() {
```

cout << "I JUST got executed!";

```
}
```

// Error

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 Refines
```

```
// Jenny Refines
```

```
// Ania Refines
```

C++ Default parameters

Default parameter value

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

It uses also default value ("Norway"):

Example ↴

```
void myfunction(String country =  
"Norway") {  
    cout << country << endl;  
}  
  
int main() {  
    myfunction("Sweden");  
    myfunction("India");  
    myfunction();  
    myfunction("USA");  
    return 0;  
}  
  
// Sweden : ("Sweden") output  
// India : ("India") output  
// Norway : ("Norway") output  
// USA : ("USA") output
```

Page No.	
Data	

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 << "Refnes."
    << age << " years old. \n";
}
```

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

```
// Liam Refnes. 3 years old.
// Terry Refnes. 14 years old.
// Ania Refnes. 30 years old.
```

C++ Return keyword :

Return Values

The void keyword, used in the previous page

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;
```

→ x and y are swapped first and then z is assigned to x .



```
int main() {
    int i, j;
    int firstNum = 10;
    int secondNum = 20;
}
```

```
cout << "Before Swap: " << firstNum << endl;
cout << endl;
cout << firstNum << secondNum << endl;
cout << endl;
```

// Call the function, which will change the values of firstNum and secondNum

```
swapNums(firstNum, secondNum);
```

```
cout << "After Swap: " << endl;
cout << firstNum << secondNum << endl;
```

```
<< endl;
return 0;
}
```

C++ pass Array to a Function:

Pass Arrays as Function Parameters

Example ↴

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

```

for (int i = 0; i < 5; i++) {
    cout << myNumbers[i] << endl;
}

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

```

C++ Function Overloading

Function Overloading : `cout << "Hello" >> f(10)`

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.

Page No.		
Date		

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; // 55
```

```
    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

creating objects that contain both data and functions.

Advantages over procedural programming :

- OOP is faster and easier to execute
- OOP provides a clear structure for the programs.
- OOP helps to keep the C++ code DRY ("Don't 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.

Class	Objects
Fruit	Apple
	Banana
	Mango



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 1

```
class MyClass { // The
  public:           // Access specifier
    int myNum;     // Attribute (int variable)
    string myString; // Attribute (string variable)
};
```

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 (intString;

Attribute (String variable)

}

```
int main () {
```

Myclass myobj; // Create an object of myclass.

// Access attributes and set values



```

myobj.myNum = 15; // myobj is forward declared
myobj.myString = "Some Text"; // myobj is forward declared
// print attribute values
cout << myobj.myNum << "\n";
cout << myobj.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";` \rightarrow `brand = "Ford"`

`carobi2.model = "Mustang";` \rightarrow `model = "Mustang"`

`carobi2.year = 1969;` \rightarrow `year = 1969`

```
// print attribute values
cout << carobi1.brand << " " << carobi1.model << " " << carobi1.year << endl;
cout << carobi2.brand << " " << carobi2.model << " " << carobi2.year << endl;
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 definition
- Outside class definition

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



Inside Example ↴

```
class MyClass {
```

class

public :

// Access

specifier

```
void myMethod () { //
```

Method / Function defined inside
the class

```
cout << "Hello World!" ;
```

```
}
```

```
int main () {
```

MyClass myobj; // Create an object
Object of MyClass

myobj.myMethod (); // Call the method

```
return 0;
```

```
}
```

Outside Example ↴

```
class MyClass {
```

class

public :

specifier

```
void myMethod ();
```

Method / Function declaration

```
}
```

Page No.		
Date		

// Method / Function definition outside the class

```
void myclass::mymethod() {
    cout << "Hello World";
}

int main() {
    myclass myobj; // Create an object of myclass
    myobj.mymethod(); // Call the function
    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 :

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 {
public:
    MyClass() { cout << "Hello World!"; }
    int main() { }
```

Page No.	
Date	

```
MyClass myobj; // Create object of MyClass (this will call constructor)
Object of MyClass (this will call constructor)
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.

For example, let's say we have a class named Car with three attributes: brand, model, and year. We can define a constructor with different parameters. Inside the constructor, we set the attributes equal to the constructor parameters (e.g., brand=x, model=y, year=z).

Example ↴

```
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 corobil ("BMW", "X5", 1999);
car corobiz ("Ford", "Mustang",
1969);
```

```
// print values
cout << corobil.brand << endl << corobil.year << endl;
corobil.model << endl;
corobil.year << endl;
cout << corobiz.brand << endl << corobiz.year << endl;
corobiz.model << endl;
corobiz.year << endl;
return 0;
};
```

C++ Access Specifiers:

Access Specifiers

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

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 << "Tut, tut! \n";

}

};

private:

// private attribute

int salary;

: encapsulation

overprotection

public:

// setter of salary with one argument

void setsalary (int s) {

 Salary = s;

}

int main () {

 Employee myobj; // -(function) constructor

 myobj.setsalary (50000);

 cout << myobj.getsalary () ;

 return 0;

}

(child) public part, private part, protected part

What is 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.

Example ↴

```
class MyClass {           // The class
public:                  // Access
    Specifier             // class members goes here
};                      // Note: here you can add constructor, destructor, etc.
```

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 myobj;
```

```
myobj.myFunction ();
```

```
myobj.myOtherFunction ();
```

```
return 0;
```

```
}
```

C++ Inheritance Access

Access Specifiers



```
cout << "some content in parent class.";
```

};

```
// Derived class (child)
```

```
class Mychild : public MyClass {
```

};

```
// Derived class (grandchild)
```

```
class MyGrandchild : public
```

Mychild {

};

```
int main () {
```

```
MyGrandchild myobj;
```

myobj.myfunction();

```
return 0;
```

}

C++ multiple Inheritance :

↳ Multiple 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 << endl;
    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
bow\n";
    }
};
```

C++ Files :

C++ Files

The `fstream` library allows us to work with files.

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



```
myobj.bonus << "n"; // writing : 29/07/2021
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
```

Example 7

```
// 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 myobj;
    myobj.setsalary (50000);
    myobj.bonus = 15000;
    cout << "Salary:" <<
    myobj.getsalary () << "\n";
    cout << "Bonus:" <<
```



Example ↴

↓ algorithm

```
#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`

Reads 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 (`<<`).

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.



Example 7

// Create a text string, which is used to output the text file to screen
 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";
}

```

things.

When an error 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++ consists 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 user 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.  
}
```