Welcome to Technical Training for Placement Preparation Masterclass by

TalentBattle

20
Hrs.

overview

What to expect.

Exactly how does this go down?

C++

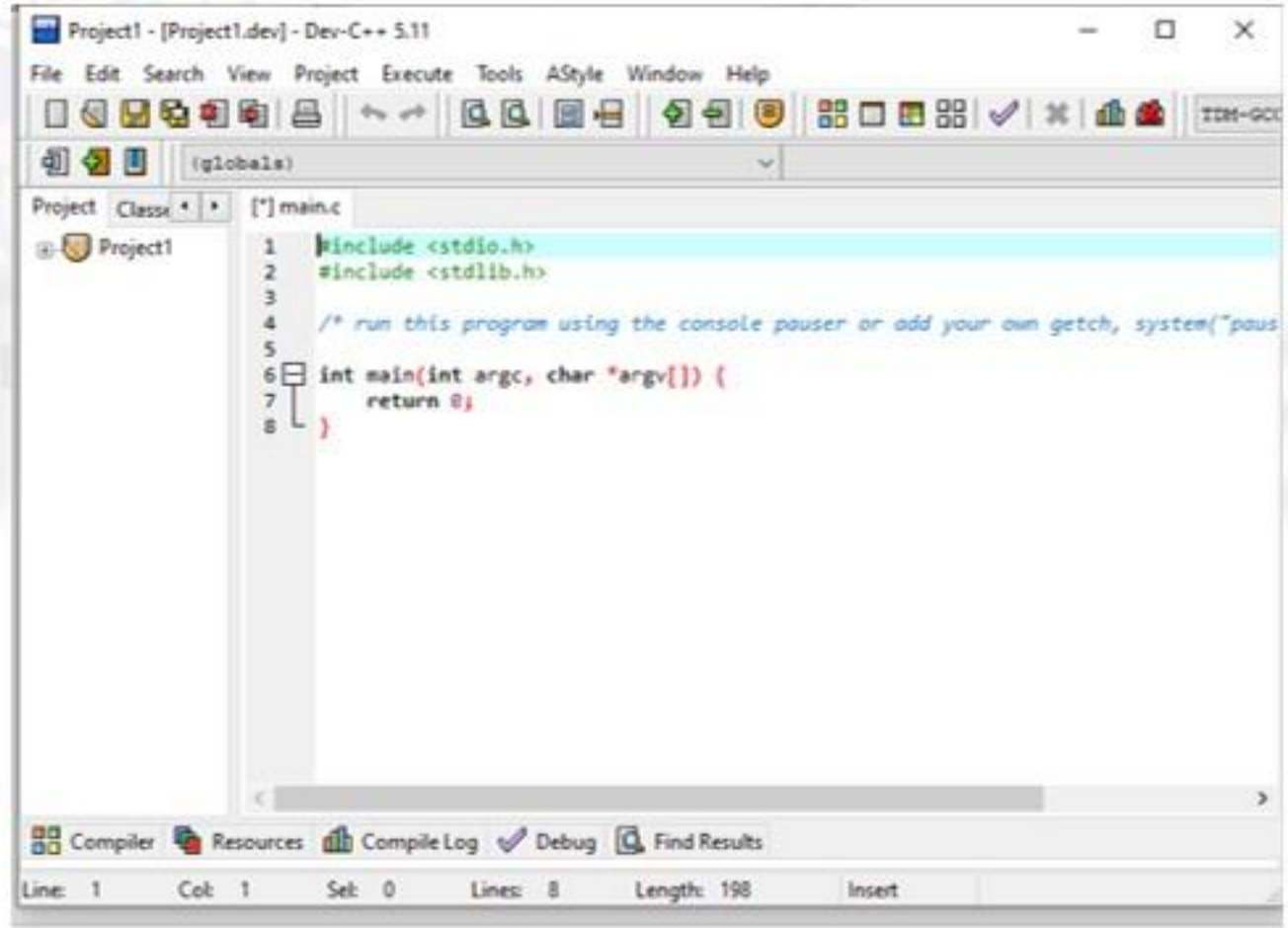
[20 Hrs.]

- In-depth training.
- Competitive programming.
- Hands-on sessions.

Writing, Executing and Fixing Code in C++

IDE : Integrated Development Environment

- Visual Studio Code
- Dev C/C++
- Code::Blocks
- Eclipse
- Netbeans
- Atom
- Sublime Text
- CodeLite
- CodeWarrior and many more…

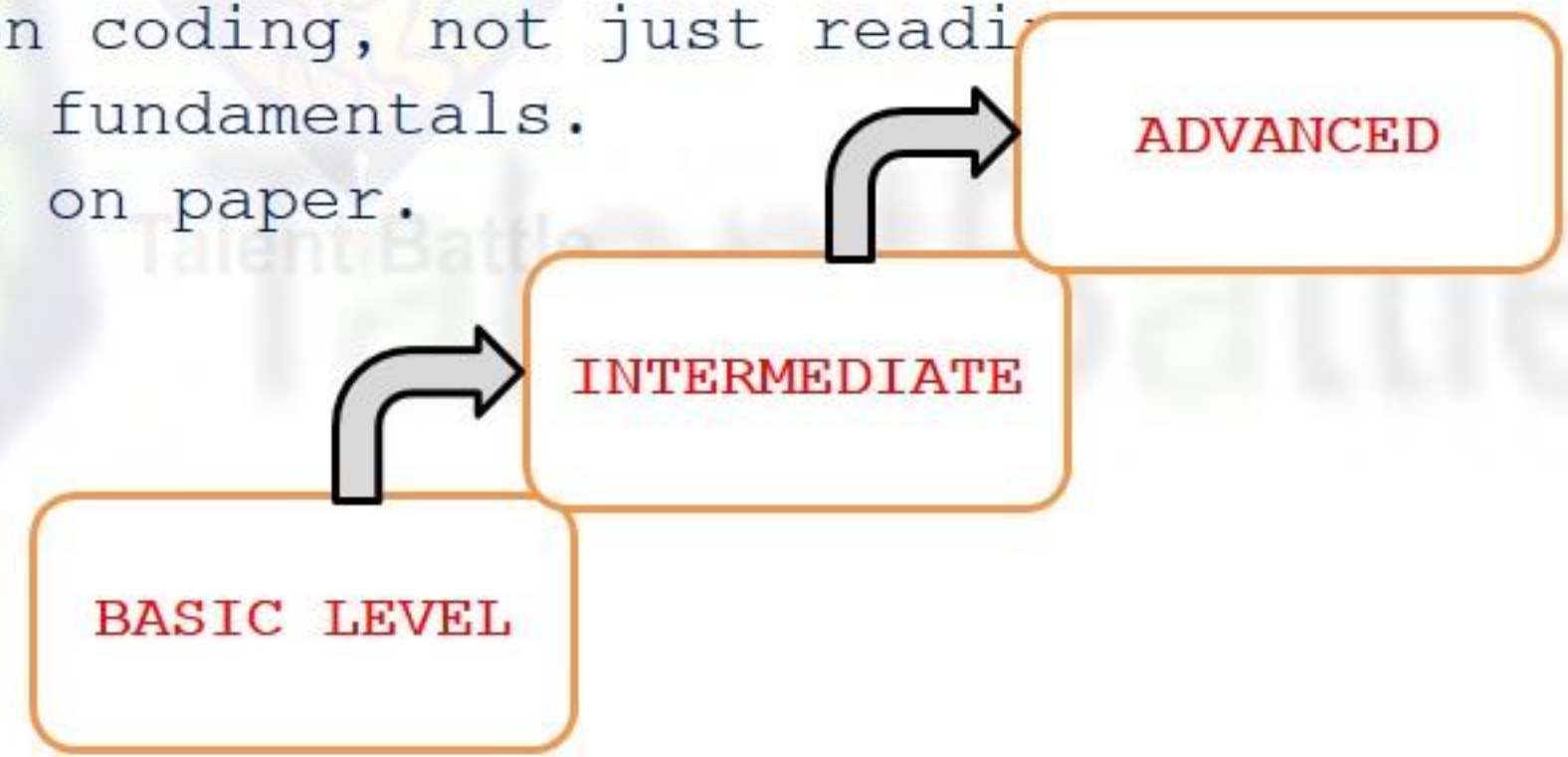


Skills you will gain…

- Algorithms
- Programming Language
- Concepts
- Problem Solving

Tips for learning CODING with no prior experience

1. Learn the basic concepts of coding first.
2. Choose the right language.
3. Learn by hands-on coding, not just readi
4. Don’t ignore the fundamentals.
5. Try writing code on paper.



What is CODING?

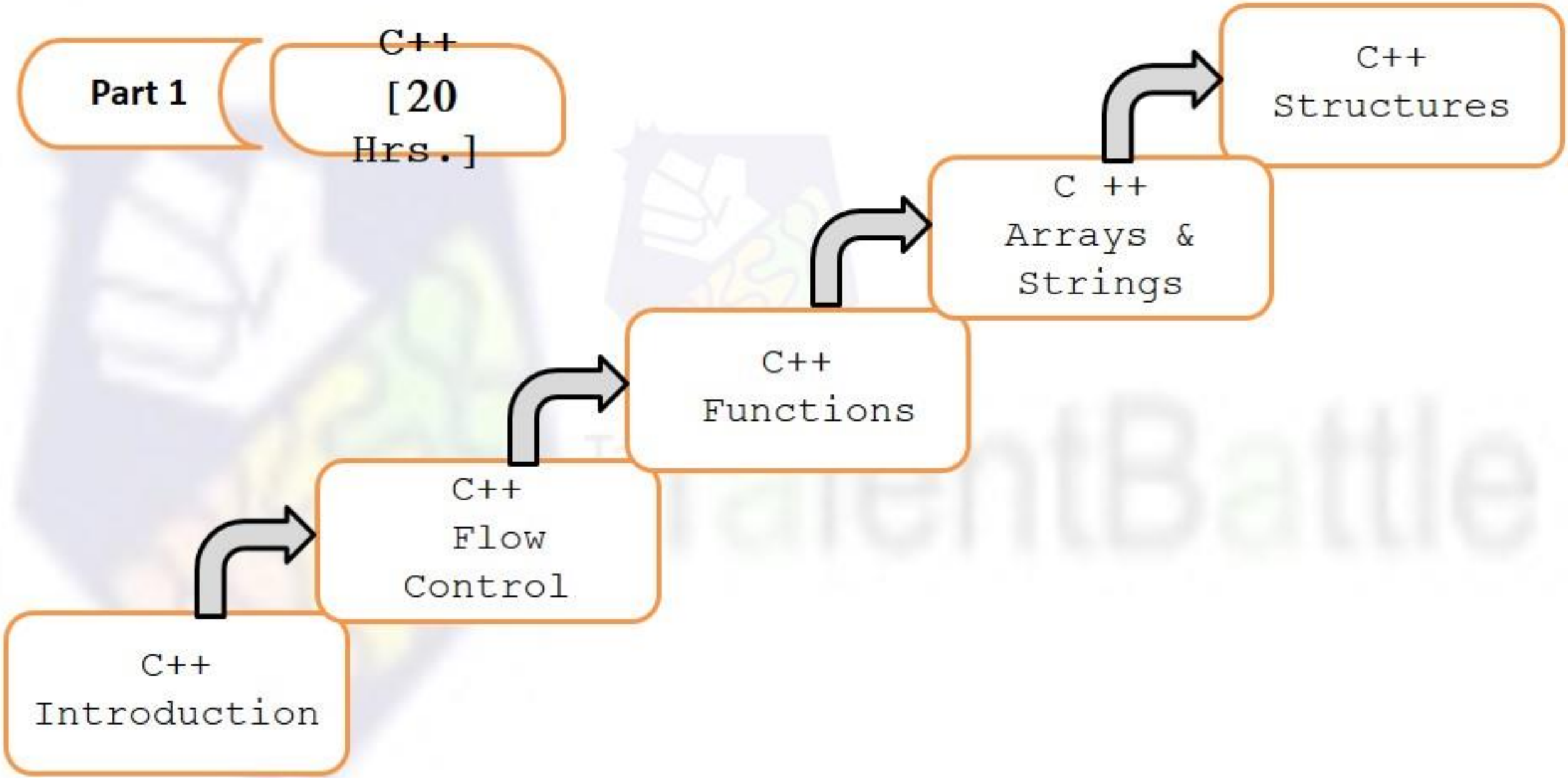
- Software Programming is also called CODING.
- CREATIVITY!!!
- Coding is a skill where you take instructions (steps) and translate it into a language that computer understands since computers do not communicate like humans.
- They communicate in a language called BINARY and it uses 0's and 1's.
- Coders write the instructions using programming language.
- Programming language translates human code into machine code known as software.

Why CODING is important?

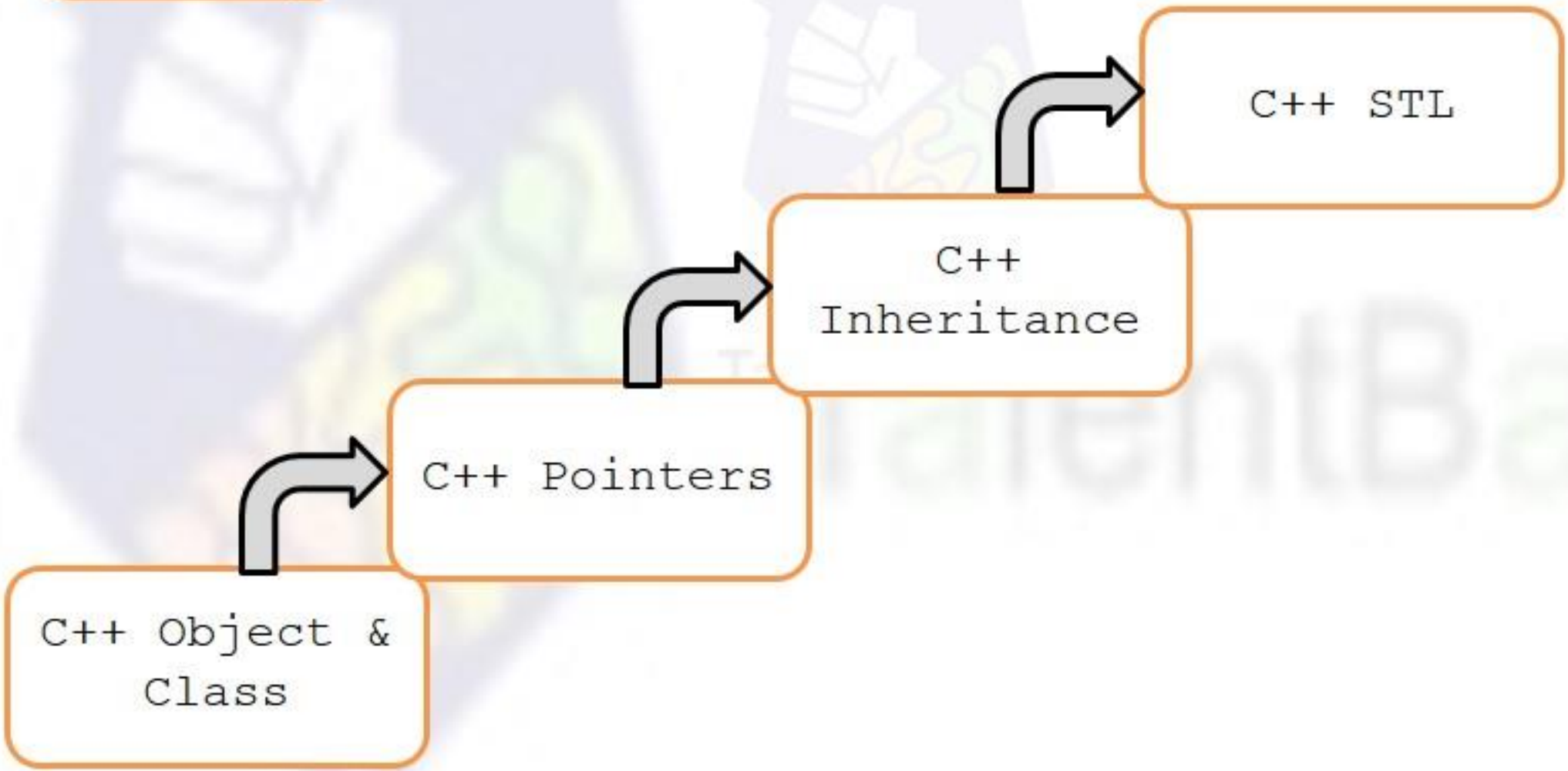
- ✓ Coding skills are good for your career prospects.
- ✓ The range of coding jobs is growing.
- ✓ More job roles now require coding skills.
- ✓ Coding skills attract higher wages.
- ✓ Learning coding skills is good for your brain.
- ✓ A career that will grow with your skills.

Part 1

C++ [20 Hrs.]



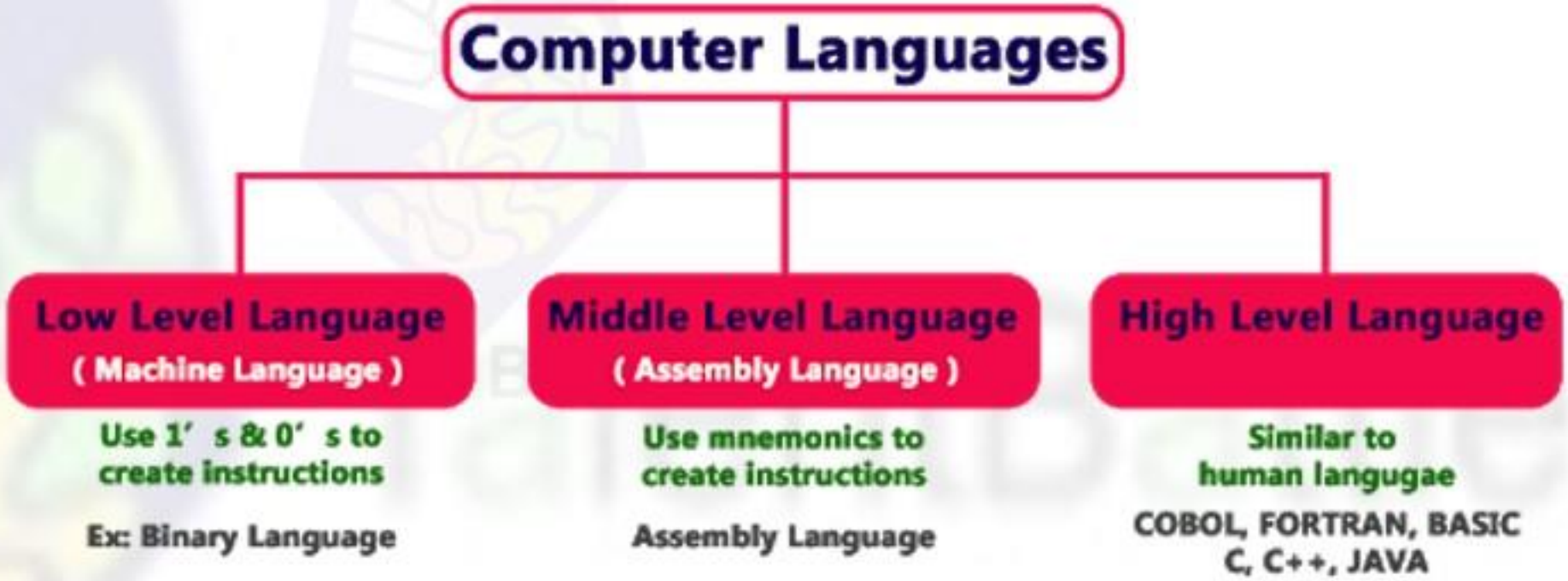
Part 2



Prerequisite

- Computer System
- Hardware
- Software
- System Software
- Application Software
- High level Language
- Machine Level Language

Programming Languages



C++
Introduction

Why Learn C++?

•C++ is used to develop games, desktop apps, operating systems, browsers, and so on because of its performance.

•After learning C++, it will be much easier to learn other programming languages like Java, Python, etc.

•C++ helps you to understand the internal architecture of a computer, how computer stores and retrieves information.

C++
Introduction

C++ Variables, Literals and Constants

C++ Variables

In programming, a variable is a container (storage area) to hold data.

To indicate the storage area, each variable should be given a unique name (identifier). For example,

```
int age = 14;
```

Here, age is a variable of the int data type, and we have assigned an integer value 14 to it.

The value of a variable can be changed, hence the name variable.

C++
Introduction

Rules for naming a variable

A variable name can only have alphabets, numbers, and the underscore _.

A variable name cannot begin with a number.

Variable names should not begin with an uppercase character.

A variable name cannot be a keyword. For example, **int** is a keyword that is used to denote integers.

A variable name can start with an underscore. However, it's not considered a good practice.

C++
Introduction

C++ Literals

Literals are data used for representing fixed values. They can be used directly in the code. For example: 1, 2.5, 'c' etc.

1. Integers

An integer is a numeric literal(associated with numbers) without any fractional or exponential part.

decimal (base 10)
octal (base 8)
hexadecimal (base 16)
For example:

Decimal: 0, -9, 22 etc
Octal: 021, 077, 033 etc
Hexadecimal: 0x7f, 0x2a, 0x521 etc

C++
Introduction

2. Floating-point Literals

A floating-point literal is a numeric literal that has either a fractional form or an exponent form. For example:

-2.0

0.0000234

C++
Introduction

3. Characters

A character literal is created by enclosing a single character inside single quotation marks. For example: 'a', 'm', 'F', '2', '}' etc.

4. Escape Sequences

Sometimes, it is necessary to use characters that cannot be typed or has special meaning in C++ programming. For example, newline (enter), tab, question mark, etc.

C++ Introduction

Escape Sequences	Characters
\b	Backspace
\f	Form feed
\n	Newline
\r	Return
\t	Horizontal tab
\v	Vertical tab
\\	Backslash
\'	Single quotation mark
\"	Double quotation mark
\?	Question mark
\0	Null Character

C++
Introduction

5. String Literals

A string literal is a sequence of characters enclosed in double-quote marks. For example:

"good"	string constant
""	null string constant
"x"	string constant having a single character
"Earth is round\n"	prints string with a newline

C++
Introduction

C++ Constants

In C++, we can create variables whose value cannot be changed. For that, we use the const keyword. Here's an example:

```
const int LIGHT_SPEED = 299792458;
LIGHT_SPEED = 2500 // Error! LIGHT_SPEED is a constant.
```

Here, we have used the keyword const to declare a constant named LIGHT_SPEED. If we try to change the value of LIGHT_SPEED, we will get an error.

A constant can also be created using the #define preprocessor directive.

C++
Introduction

C++ Data Types

In C++, data types are declarations for variables. This determines the type and size of data associated with variables. For example,

```
int age = 13;
```

Here, age is a variable of type int. Meaning, the variable can only store integers of either 2 or 4 bytes.

C++ Introduction

C++ Fundamental Data Types

Data Type	Meaning	Size (in Bytes)
int	Integer	2 or 4
float	Floating-point	4
double	Double Floating-point	8
char	Character	1
bool	Boolean	1
void	Empty	0

C++
Introduction

1. C++ int

The int keyword is used to indicate integers.

Its size is usually 4 bytes. Meaning, it can store values from -

2147483648 to 2147483647.

For example,

int salary = 85000;

C++
Introduction

2. C++ float and double

float and double are used to store floating-point numbers (decimals and exponentials).

The size of float is 4 bytes and the size of double is 8 bytes. Hence, double has two times the precision of float. To learn more, visit C++ float and double.

For example,

```
float area = 64.74;

double volume = 134.64534;
```

C++
Introduction

3. C++ char

Keyword char is used for characters.

Its size is 1 byte.

Characters in C++ are enclosed inside single quotes ' '.

For example,

```
char test = 'h';
```

C++
Introduction

4. C++ bool

The bool data type has one of two possible values: true or false.

Booleans are used in conditional statements and loops (which we will learn in later chapters).

For example,

```
bool cond = false;
```

C++
Introduction

5. C++ void

The void keyword indicates an absence of data. It means "nothing" or "no value".

We will use void when we learn about functions and pointers.

Note: We cannot declare variables of the void type.

C++
Introduction

C++ Type Modifiers

We can further modify some of the fundamental data types by using type modifiers. There are 4 type modifiers in C++. They are:

signed
unsigned
short
long

We can modify the following data types with the above modifiers:

int
double
char

C++
Introduction

Derived Data Types

Data types that are derived from fundamental data types are derived types.

For example: arrays, pointers, function types, structures, etc.

C++
Introduction

C++ Basic Input/Output

C++ Output

In C++, cout sends formatted output to standard output devices, such as the screen. We use the cout object along with the << operator for displaying output.

cout is an object that prints the string inside quotation marks " ".

It is followed by the << operator.

C++
Introduction

C++ Input

In C++, cin takes formatted input from standard input devices such as the keyboard.

We use the cin object along with the >> operator for taking input.

Note: If we don't include the using namespace std; statement, we need to use std::cin instead of cin.

C++
Introduction

C++ Type Conversion

C++ allows us to convert data of one type to that of another. This is known as type conversion.

There are two types of type conversion in C++.

- **Implicit Conversion**
- **Explicit Conversion (also known as Type Casting)**

C++
Introduction

Implicit Type Conversion

The type conversion that is done automatically done by the compiler is known as implicit type conversion. This type of conversion is also known as automatic conversion.

```
// Working of implicit type-conversion

#include <iostream>
using namespace std;
int main() {
  // assigning an int value to num_int
  int num_int = 9;
  // declaring a double type variable
  double num_double;
  // implicit conversion
  // assigning int value to a double variable
  num_double = num_int;
  cout << "num_int = " << num_int << endl;
  cout << "num_double = " << num_double << endl;
  return 0;
}
```

C++
Introduction

C++ Explicit Conversion

When the user manually changes data from one type to another, this is known as **explicit conversion**. This type of conversion is also known as **type casting**.

There are three major ways in which we can use explicit conversion in C++. They are:

1.C-style type casting (also known as **cast notation**)

2.Function notation (also known as **old C++ style type casting**)

3.Type conversion operators

C++
Introduction

C-style Type Casting

As the name suggests, this type of casting is favored by the C programming language. It is also known as cast notation.

The syntax for this style is:

```
(data_type)expression;
```

For example,

```
// initializing int variable
int num_int = 26;
// declaring double variable
double num_double;
// converting from int to double
num_double = (double)num_int;
```

C++
Introduction

Function-style Casting

We can also use the function like notation to cast data from one type to another.

The syntax for this style is:

```
data_type(expression);
```

For example,

```
// initializing int variable
int num_int = 26;
// declaring double variable
double num_double;
// converting from int to double
num_double = double(num_int);
```

C++
Introduction

C++ Operators

Operators are symbols that perform operations on variables and values. For example, + is an operator used for addition, while - is an operator used for subtraction.

Operators in C++ can be classified into 6 types:

- Arithmetic Operators
- Assignment Operators
- Relational Operators
- Logical Operators
- Bitwise Operators
- Other Operators

C++
Introduction

1. C++ Arithmetic Operators

Arithmetic operators are used to perform arithmetic operations on variables and data. For example,

```
a + b;
```

Here, the + operator is used to add two variables a and b. Similarly there are various other arithmetic operators in C++.

Operator	Operation
+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Modulo Operation (Remainder after division)

C++
Introduction

Increment and Decrement Operators

C++ also provides increment and decrement operators: ++ and -- respectively.

++ increases the value of the operand by 1
-- decreases it by 1

C++ Introduction

2. C++ Assignment Operators

In C++, assignment operators are used to assign values to variables.

For example,

```
// assign 5 to a
a = 5;
```

Here, we have assigned a value of 5 to the variable a.

Operator	Example	Equivalent to
=	a = b;	a = b;
+=	a += b;	a = a + b;
-=	a -= b;	a = a - b;
*=	a *= b;	a = a * b;
/=	a /= b;	a = a / b;
%=	a %= b;	a = a % b;

C++
Introduction

3. C++ Relational Operators

A relational operator is used to check the relationship between two operands. For example,

```
// checks if a is greater than b
a > b;
```

Here, > is a relational operator. It checks if a is greater than b or not.

If the relation is true, it returns 1 whereas if the relation is false, it returns 0.

Operator	Meaning	Example
==	Is Equal To	3 == 5 gives us false
!=	Not Equal To	3 != 5 gives us true
>	Greater Than	3 > 5 gives us false
<	Less Than	3 < 5 gives us true
>=	Greater Than or Equal To	3 >= 5 give us false
<=	Less Than or Equal To	3 <= 5 gives us true

C++ Introduction

4. C++ Logical Operators

Logical operators are used to check whether an expression is true or false. If the expression is true, it returns 1 whereas if the expression is false, it returns 0.

Operator	Example	Meaning
&&	expression1 && expression2	Logical AND.
True only if all the operands are true.		
||	expression1 || expression2	Logical OR.
True if at least one of the operands is true.		
!	!expression	Logical NOT.
True only if the operand is false.		

In C++, logical operators are commonly used in decision making.

C++
Introduction

5. C++ Bitwise Operators

In C++, bitwise operators are used to perform operations on individual bits. They can only be used alongside char and int data types.

Operator	Description
&	Binary AND
|	Binary OR
^	Binary XOR
~	Binary One's Complement
<<	Binary Shift Left
>>	Binary Shift Right

C++
Introduction

C++ Comments

C++ comments are hints that a programmer can add to make their code easier to read and understand. They are completely ignored by C++ compilers.

There are two ways to add comments to code:

// - Single Line Comments

/* */ -Multi-line Comments

C++
Introduction

Practice Examples:

- C++ "Hello, World!" Program
- C++ Program to Print Number Entered by User
- C++ Program to Add Two Numbers
- C++ Program to Find Quotient and Remainder
- C++ Program to Find Size of int, float, double and char in Your System
- C++ Program to Swap Two Numbers
- C++ Program to Find ASCII Value of a Character
- C++ Program to Multiply two Numbers

C++ Flow Control

C++ if, if...else and Nested if...else

In computer programming, we use the if statement to run a block code only when a certain condition is met.

For example, assigning grades (A, B, C) based on marks obtained by a student.

if the percentage is above 90, assign grade A
if the percentage is above 75, assign grade B
if the percentage is above 65, assign grade C

C++ Flow Control

There are three forms of if...else statements in C++.

if statement
if...else statement
if...else if...else statement

C++ Flow Control

C++ if Statement

The syntax of the if statement is:

```
if (condition) {
  // body of if statement
}
```

The if statement evaluates the condition inside the parentheses ( ).

If the condition evaluates to true, the code inside the body of if is executed.

If the condition evaluates to false, the code inside the body of if is skipped.

Note: The code inside { } is the body of the if statement.

C++ Flow Control

Condition is true

```
int number = 5;

if (number > 0) {
    // code
}

// code after if
```

Condition is false

```
int number = 5;

if (number < 0) {
    // code
}

// code after if
```

C++ Flow Control

C++ if...else

The if statement can have an optional else clause. Its syntax is:

```
if (condition) {
   // block of code if condition is true
}
else {
   // block of code if condition is false
}
```

The if..else statement evaluates the condition inside the parenthesis.

C++ Flow Control

Condition is true

```
int number = 5;

if (number > 0) {
    // code
}
else {
    // code
}

// code after if...else
```

Condition is false

```
int number = 5;

if (number < 0) {
    // code
}
else {
    // code
}

// code after if...else
```

C++ Flow Control

C++ if...else...else if statement

The if...else statement is used to execute a block of code among two alternatives. However, if we need to make a choice between more than two alternatives, we use the if...else if...else statement.

The syntax of the if...else if...else statement is:

```
if (condition1) {
   // code block 1
}
else if (condition2){
   // code block 2
}
else {
   // code block 3
}
```

C++ Flow Control

1st Condition is true

```
int number = 2;
if (number > 0) {
    // code
}
else if (number == 0){
    // code
}
else {
    //code
}

//code after if
```

2nd Condition is true

```
int number = 0;
if (number > 0) {
    // code
}
else if (number == 0){
    // code
}
else {
    //code
}

//code after if
```

All Conditions are false

```
int number = -2;
if (number > 0) {
    // code
}
else if (number == 0){
    // code
}
else {
    //code
}

//code after if
```

C++ Flow Control

C++ Nested if...else

Sometimes, we need to use an if statement inside another if statement. This is known as nested if statement.

Think of it as multiple layers of if statements. There is a first, outer if statement, and inside it is another, inner if statement. Its syntax is:

```
// outer if statement
if (condition1) {
   // statements
   // inner if statement
   if (condition2) {
      // statements
   }
}
```

Notes:

We can add else and else if statements to the inner if statement as required.

The inner if statement can also be inserted inside the outer else or else if statements (if they exist).

We can nest multiple layers of if statements.

C++ Flow Control

Example : C++ Nested if

// C++ program to find if an integer is even or odd or neither (0) using nested if statements

C++ Flow Control

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

int main() {
   int num;
    cout << "Enter an integer: ";
    cin >> num;
   // outer if condition
   if (num != 0) {
      // inner if condition
      if ((num % 2) == 0) {
         cout << "The number is even." << endl;
      }
       // inner else condition
      else {
         cout << "The number is odd." << endl;
      }
   }
   // outer else condition
   else {
      cout << "The number is 0 and it is neither even nor odd." << endl;
   }
   cout << "This line is always printed." << endl;
}
```

C++ Flow Control

Check out these examples to learn more:

- C++ Program to Check Whether Number is Even or Odd
- C++ Program to Check Whether a Character is Vowel or Consonant.
- C++ Program to Find Largest Number Among Three Numbers

C++ Flow Control

C++ for Loop

In computer programming, loops are used to repeat a block of code.

For example, let's say we want to show a message 100 times. Then instead of writing the print statement 100 times, we can use a loop.

That was just a simple example; we can achieve much more efficiency and sophistication in our programs by making effective use of loops.

There are 3 types of loops in C++.

- for loop
- while loop
- do...while loop

C++ Flow Control

C++ for loop

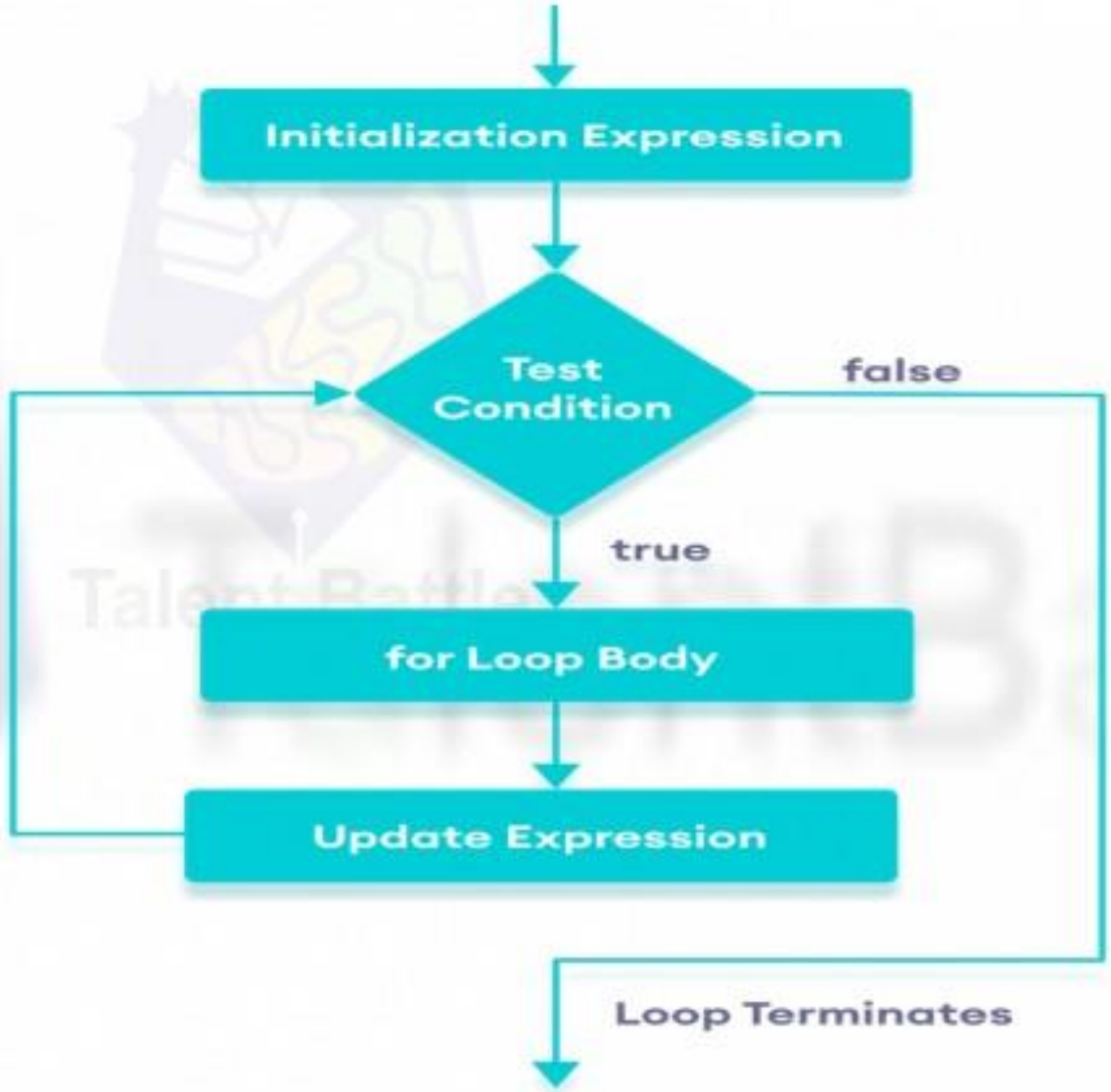
The syntax of for-loop is:

```
for (initialization; condition; update) {
   // body of-loop
}
```

Here,

initialization - initializes variables and is executed only once
condition - if true, the body of for loop is executed
if false, the for loop is terminated
update - updates the value of initialized variables and again checks the condition

C++ Flow Control



C++ Flow Control

Example : **Range Based for Loop**

```
#include <iostream>

using namespace std;

int main() {

   int num_array[] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};

   for (int n : num_array) {
      cout << n << " ";
   }

   return 0;
}
```

C++ Flow Control

Check out these examples to learn more:

- C++ Program to Calculate Sum of Natural Numbers
- C++ Program to Find Factorial
- C++ Program to Generate Multiplication Table

C++ Flow Control

C++ while and do...while Loop

C++ while Loop

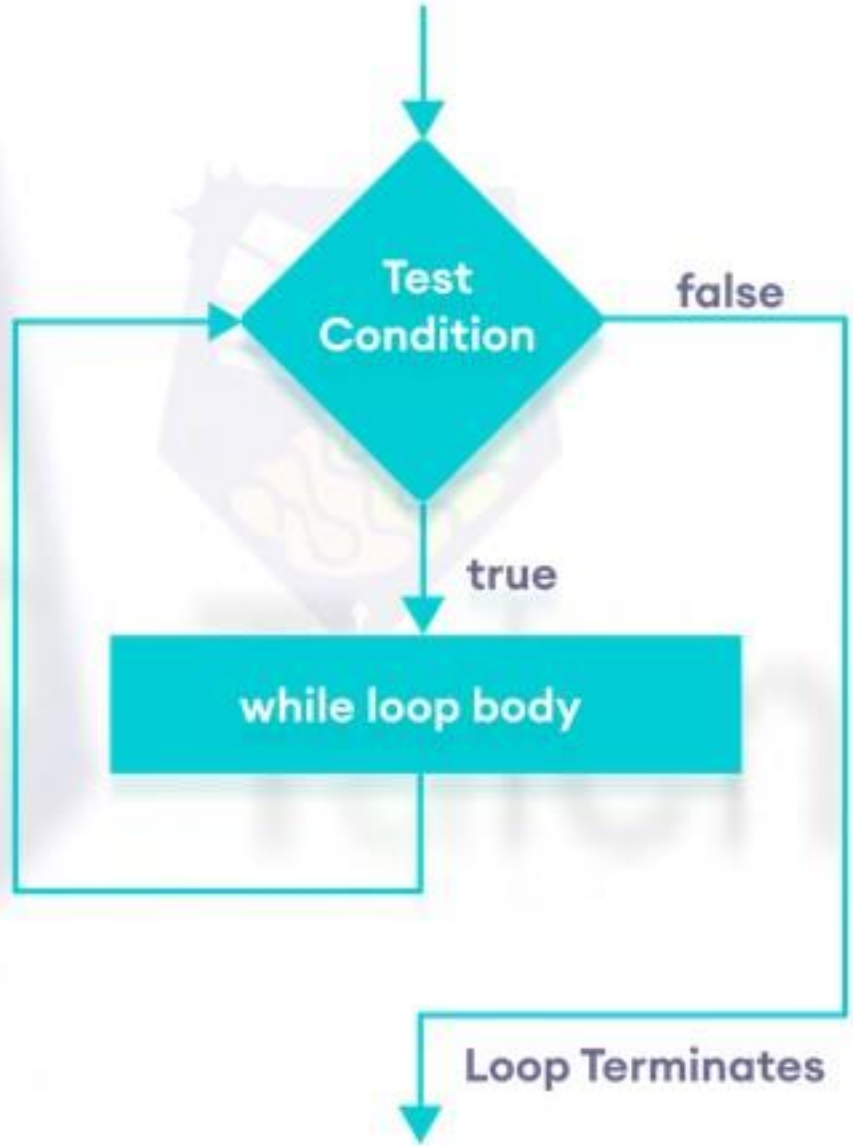
The syntax of the while loop is:

```
while (condition) {
   // body of the loop
}
```

Here,

A while loop evaluates the condition
If the condition evaluates to true, the code inside the while loop is executed.
The condition is evaluated again.
This process continues until the condition is false.
When the condition evaluates to false, the loop terminates.

C++ Flow Control



C++ Flow Control

C++ do...while Loop

The do...while loop is a variant of the while loop with one important difference: the body of do...while loop is executed once before the condition is checked.

Its syntax is:

```
do {
  // body of loop;
}
while (condition);
```

Here,

The body of the loop is executed at first. Then the condition is evaluated.

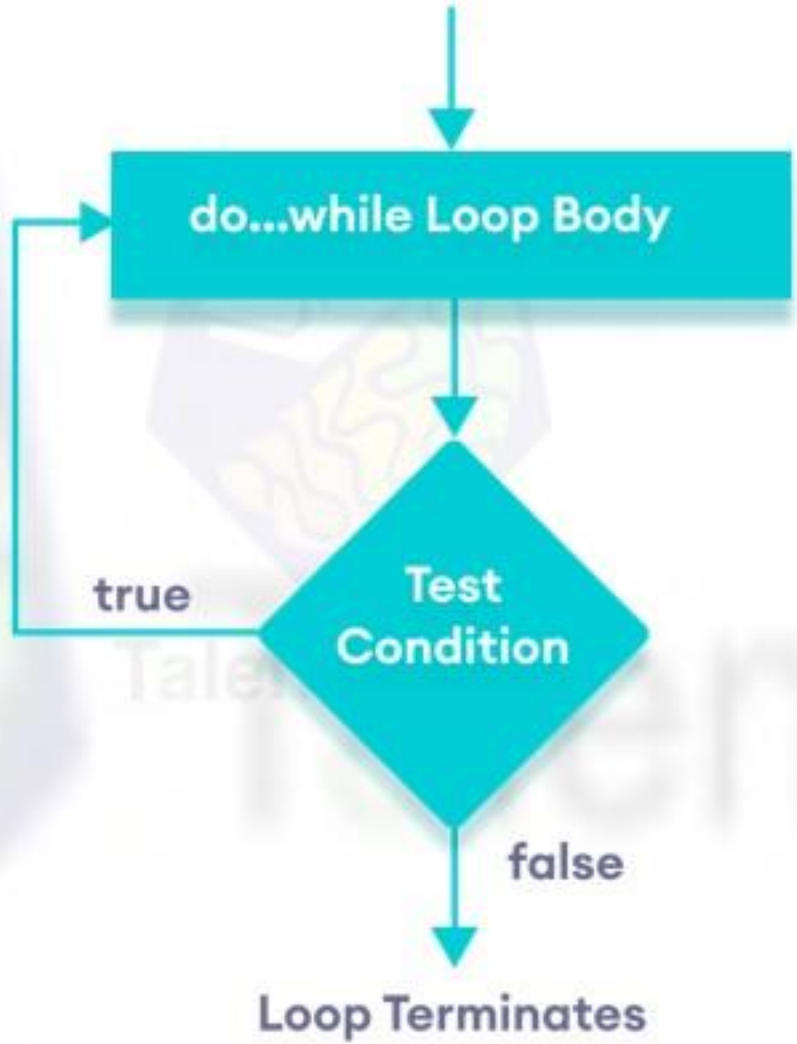
If the condition evaluates to true, the body of the loop inside the do statement is executed again.

The condition is evaluated once again.

If the condition evaluates to true, the body of the loop inside the do statement is executed again.

This process continues until the condition evaluates to false. Then the loop stops.

C++ Flow Control



C++ Flow Control

Check out these examples to learn more:

- C++ Program to Display Fibonacci Series
- C++ Program to Find GCD
- C++ Program to Find LCM

C++ Flow Control

C++ break Statement

In C++, the break statement terminates the loop when it is encountered.

The syntax of the break statement is:

```
break;
```

The break statement is also used with the switch statement.

C++ Flow Control

```
for (init; condition; update) {
    // code
    if (condition to break) {
        break;
    }
    // code
}
```

```
while (condition) {
    // code
    if (condition to break) {
        break;
    }
    // code
}
```

C++ Flow Control

C++ continue Statement

In computer programming, the continue statement is used to skip the current iteration of the loop and the control of the program goes to the next iteration.

The syntax of the continue statement is:

```
continue;
```

C++ Flow Control

```
for (init; condition; update) {
    // code
    if (condition to break) {
        continue;
    }
    // code
}
```

```
while (condition) {
    // code
    if (condition to break) {
        continue;
    }
    // code
}
```

C++ Flow Control

```
// program to print the value of i

#include <iostream>
using namespace std;

int main() {
    for (int i = 1; i <= 5; i++) {
        // condition to continue
        if (i == 3) {
            continue;
        }

        cout << i << endl;
    }

    return 0;
}
```

C++ Flow Control

Note: The break statement terminates the loop entirely. However, the continue statement only skips the current iteration.

C++ Flow Control

C++ switch..case Statement

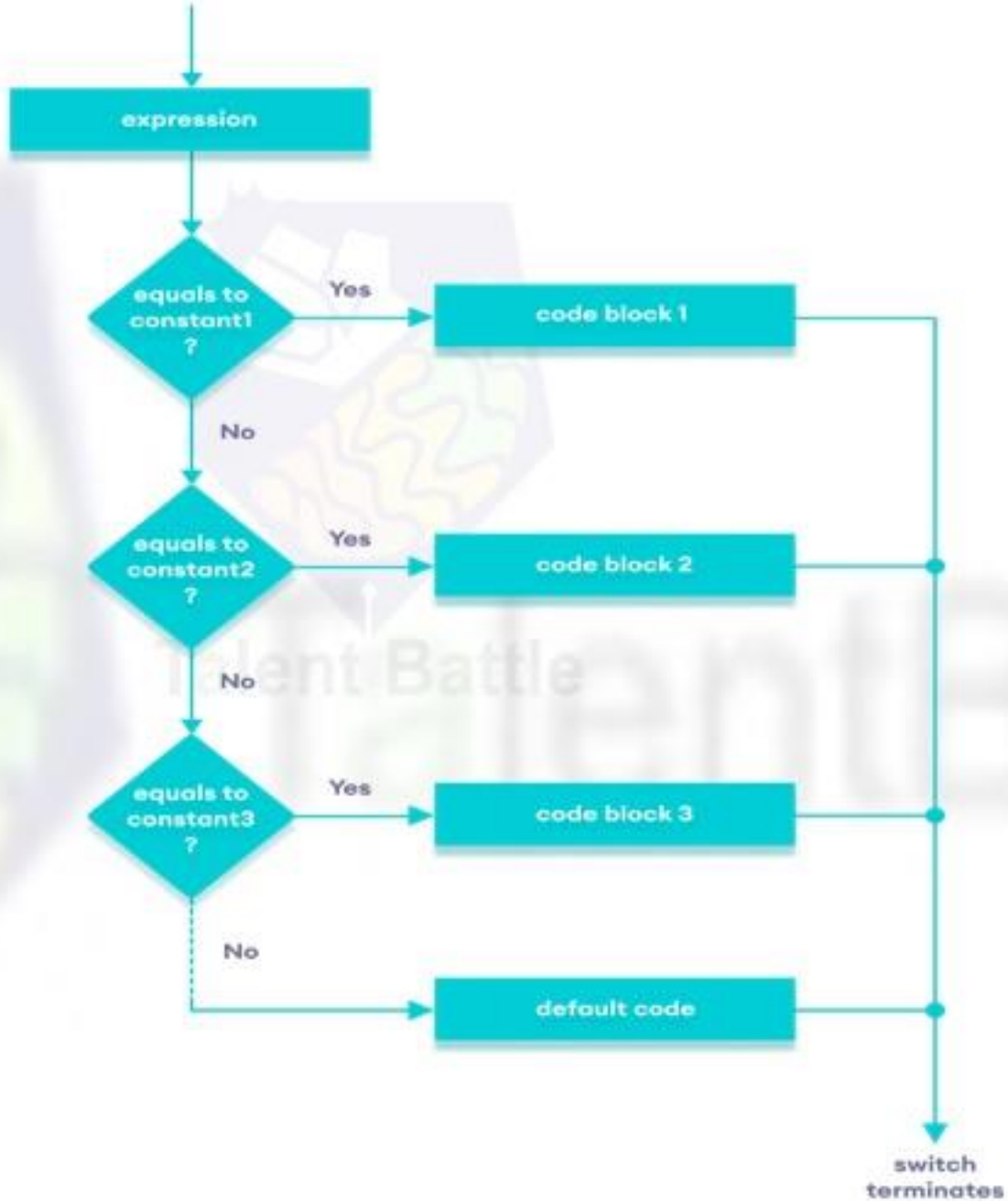
The switch statement allows us to execute a block of code among many alternatives.

The syntax of the switch statement in C++ is:

```
switch (expression) {
  case constant1:
    // code to be executed if
    // expression is equal to constant1;
    break;

  case constant2:
    // code to be executed if
    // expression is equal to constant2;
    break;
    .
    .
    .
  default:
    // code to be executed if
    // expression doesn't match any constant
}
```

C++ Flow Control



C++ Flow Control

Example: Create a Calculator using the switch Statement

Output :

```
Enter an operator (+, -, *, /): +
Enter two numbers:
2.3
4.5
2.3 + 4.5 = 6.8
```

C++ Flow Control

```
// Program to build a simple calculator using switch Statement
#include <iostream>
using namespace std;

int main() {
  char oper;
  float num1, num2;
  cout << "Enter an operator (+, -, *, /): ";
  cin >> oper;
  cout << "Enter two numbers: " << endl;
  cin >> num1 >> num2;

  switch (oper) {
    case '+':
      cout << num1 << " + " << num2 << " = " << num1 + num2;
      break;
    case '-':
      cout << num1 << " - " << num2 << " = " << num1 - num2;
      break;
    case '*':
      cout << num1 << " * " << num2 << " = " << num1 * num2;
      break;
    case '/':
      cout << num1 << " / " << num2 << " = " << num1 / num2;
      break;
    default:
      // operator is doesn't match any case constant (+, -, *, /)
      cout << "Error! The operator is not correct";
      break;
  }

  return 0;
}
```

C++ Flow Control

Notice that the break statement is used inside each case block.

This terminates the switch statement.

If the break statement is not used, all cases after the correct case are executed.

C++ Flow Control

C++ goto Statement

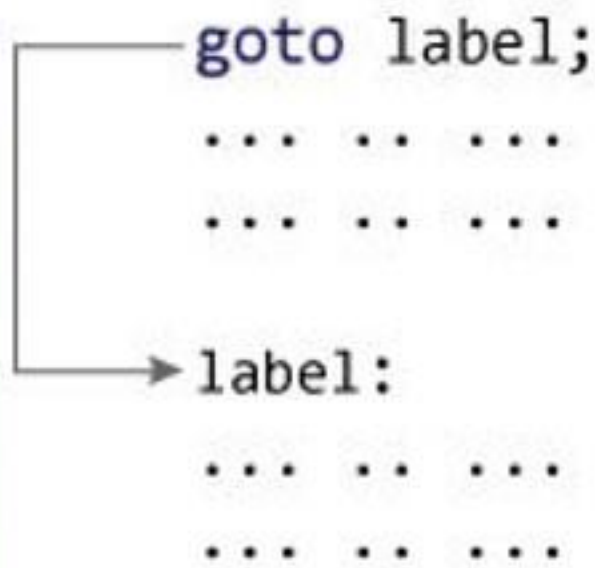
In C++ programming, goto statement is used for altering the normal sequence of program execution by transferring control to some other part of the program.

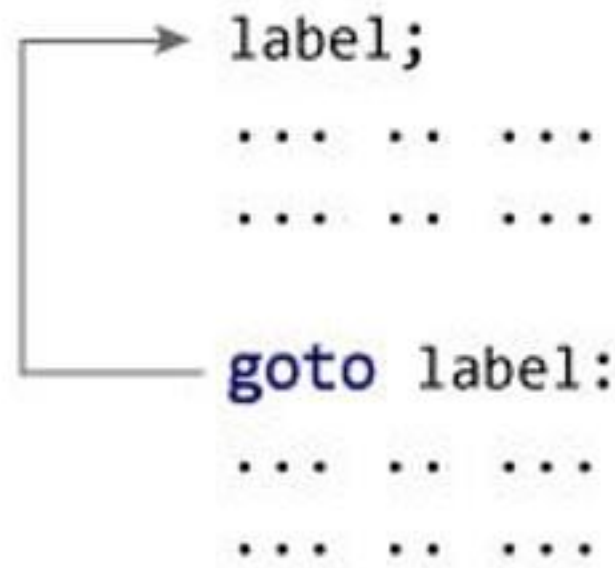
Syntax of goto Statement

```
goto label;
... .. ...
... .. ...
... .. ...
label:
statement;
... .. ...
```

In the syntax above, label is an identifier. When goto label; is encountered, the control of program jumps to label: and executes the code below it.

C++ Flow Control





C++ Flow Control

```
// This program calculates the average of numbers entered by the user.
// If the user enters a negative number, it ignores the number and
// calculates the average number entered before it.

# include <iostream>
using namespace std;

int main()
{
   float num, average, sum = 0.0;
   int i, n;
   cout << "Maximum number of inputs: ";
   cin >> n;
   for(i = 1; i <= n; ++i)
   {
      cout << "Enter n" << i << ": ";
      cin >> num;
      if(num < 0.0)
      {
        // Control of the program move to jump:
         goto jump;
      }
      sum += num;
   }
jump:
   average = sum / (i - 1);
   cout << "\nAverage = " << average;
   return 0;
}
```

C++ Flow Control

Reason to Avoid goto Statement

The goto statement gives the power to jump to any part of a program but, makes the logic of the program complex and tangled.

In modern programming, the goto statement is considered a harmful construct and a bad programming practice.

The goto statement can be replaced in most of C++ program with the use of break and continue statements.

C++ Flow Control

C++ Program to Check Whether Number is Even or Odd
C++ Program to Check Whether a character is Vowel or Consonant.
C++ Program to Find Largest Number Among Three Numbers
C++ Program to Find All Roots of a Quadratic Equation
C++ Program to Calculate Sum of Natural Numbers
C++ Program to Check Leap Year
C++ Program to Find Factorial
C++ Program to Generate Multiplication Table
C++ Program to Display Fibonacci Series
C++ Program to Find GCD
C++ Program to Find LCM
C++ Program to Reverse a Number
C++ Program to Calculate Power of a Number
C++ Program to Check Whether a Number is Palindrome or Not
C++ Program to Check Whether a Number is Prime or Not
C++ Program to Display Prime Numbers Between Two Intervals
C++ Program to Check Armstrong Number
C++ Program to Display Armstrong Number Between Two Intervals
C++ Program to Display Factors of a Number
C++ Programs To Create Pyramid and Pattern
C++ Program to Make a Simple Calculator to Add, Subtract, Multiply or Divide Using switch...case