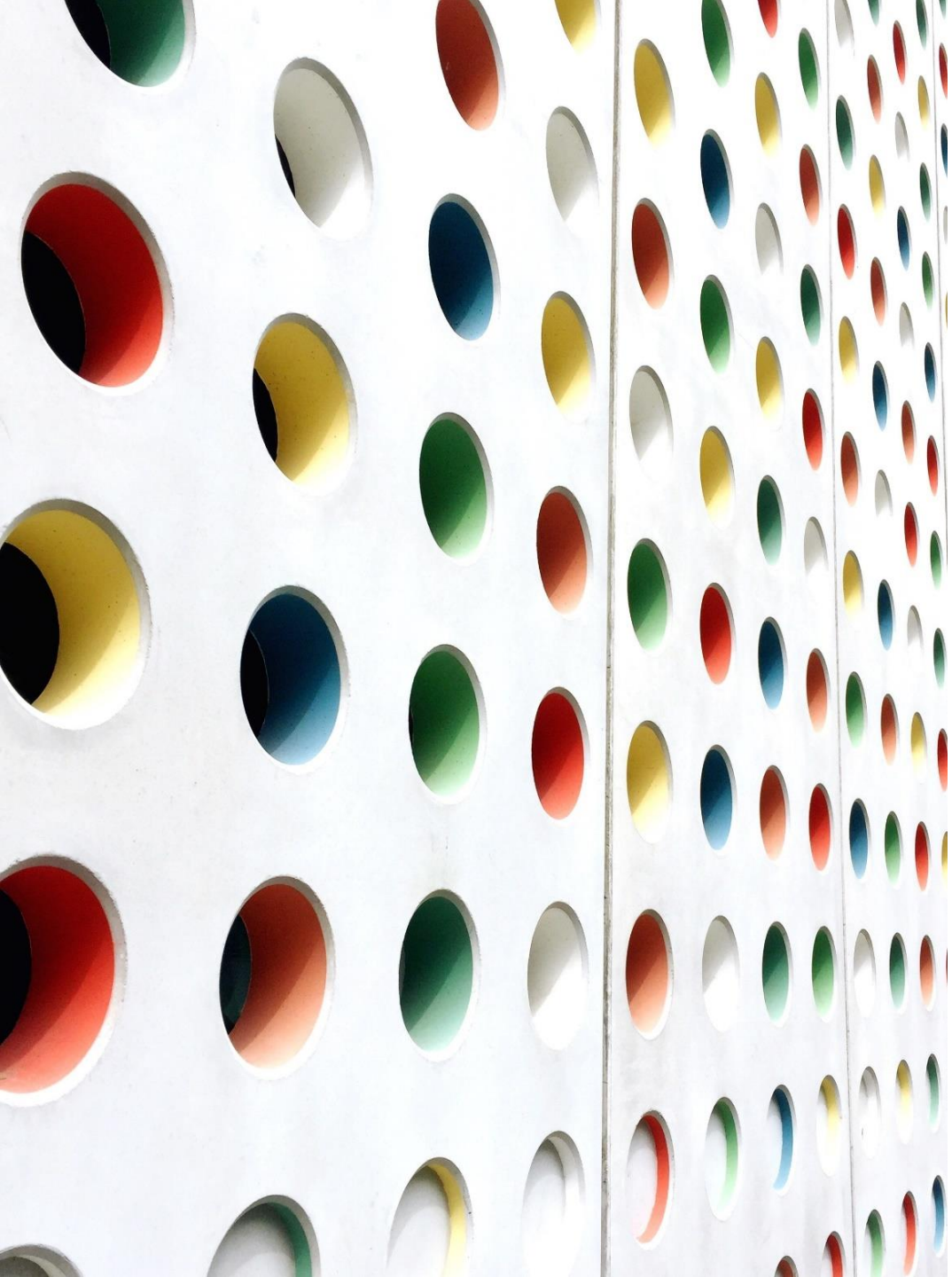


Chapter One

Basics of Programming

Prepared by: | *Sinodos G (MSc.)*

Part: One

Introduction

- What is Programming?
- Types of programming languages

Programming Language

- Computer language that is used by programmers (developers) to communicate with computers.
- Set of instructions written in any specific language (C, C++, C++, Python, PHP, JS, …) to perform a specific task.
- Mainly used:-
  - To design and implement different types of software's [desktop applications, Enterprise, websites, and mobile applications]
  - .

Programming Paradigm's

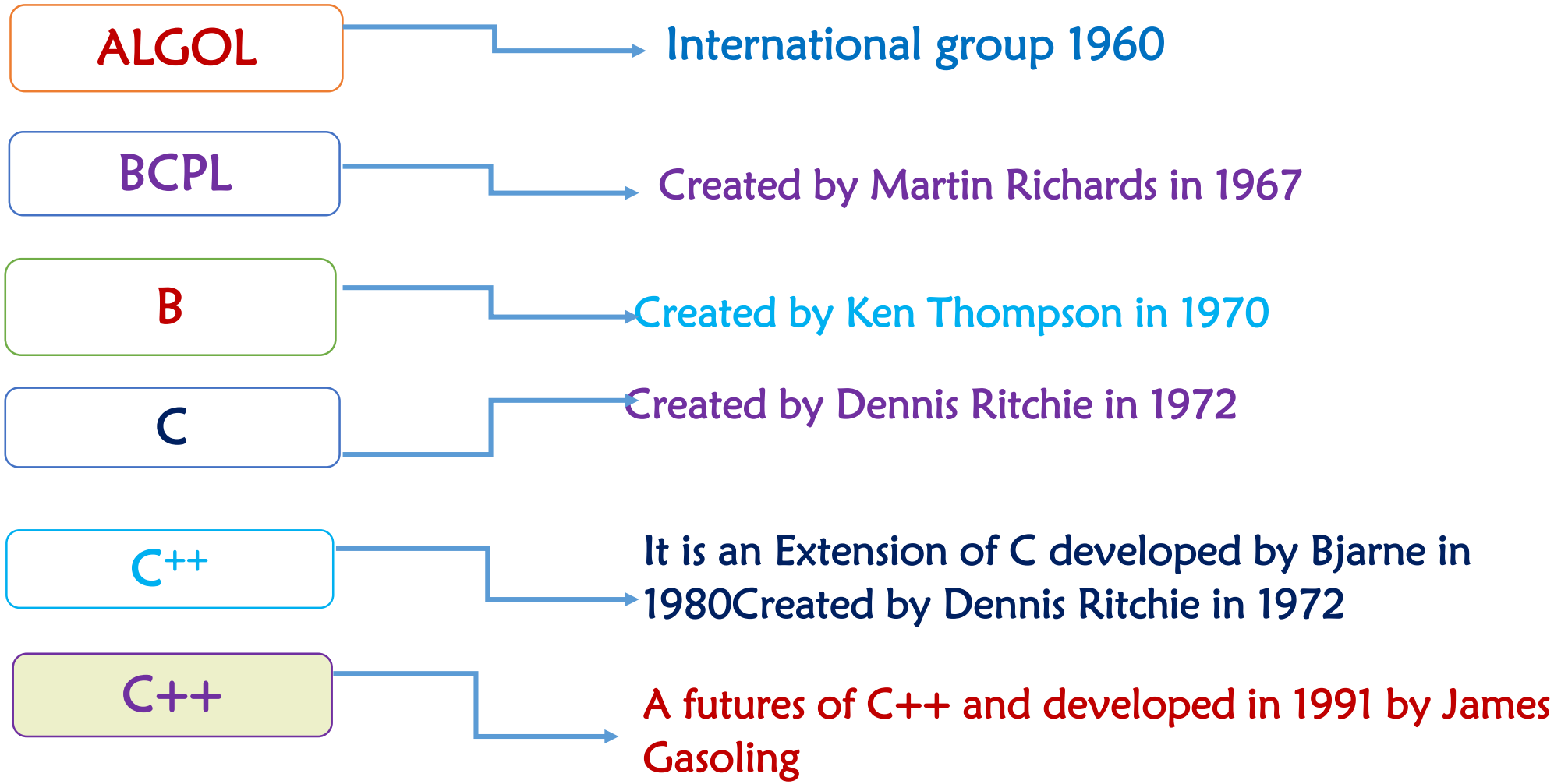
- paradigm's a style, or "way," of programming.
- some languages make it easy to write in some paradigms but not others.
- a way to classify programming languages based on their style and approach to solving problems.
- each paradigm consists of certain structures, features, and opinions about how common programming problems should be tackled.

Algorithm

- a blueprint for writing code to automate processes and achieve desired results.
- a step-by-step procedure or a set of well-defined instructions for solving a specific problem or performing a task.

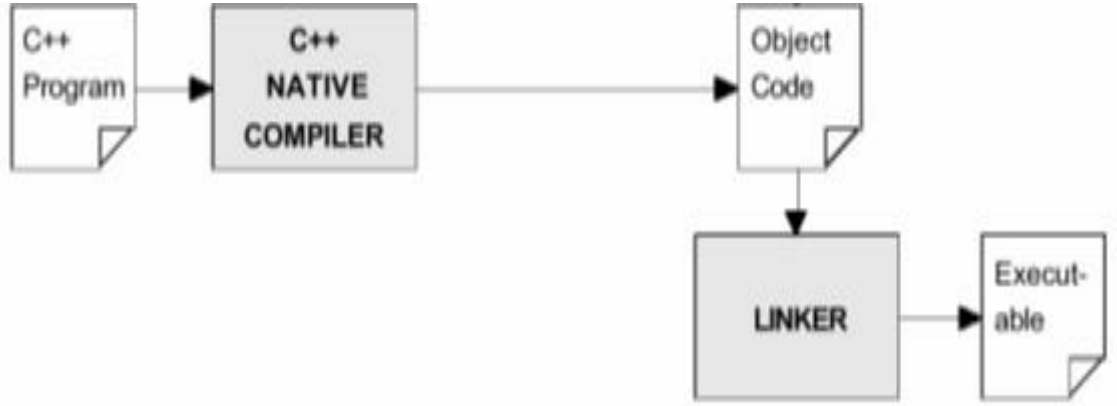
- Algorithms are the heart of computer programming.

History of Programming



C++ IDE

- Stands for Integrated Development Environment
- It is a software application that provides compressive facilities of computer programs for software dev.t.
- A software suite that consolidates basic tools required to write and test software.
- C++ IDE it consists of the following components:-
  - Editors
  - Preprocess
  - Compiling
  - Linking
  - Loading



Cont'd ...

- Some examples of popular C++ IdE's are:-
  - Jet brains
  - Eclipse
  - Code:: Blocks
  - NetBeans
  - Dev C++
  - Visual studio C++
  - Turbo C++
  - Quincy
  - …

C++ Program Structure

- Follow the following Simple C++ program

```
//Simple c++ program

#include<iostream>
using namespace std;

int main()
 {
  cout<<"Hello Ethiopia"<<endl;
  return 0;
 }
```

Output

```
Hello Ethiopia
```

C++ Identifiers

- a name used to identify a variable, function, class, module, or any other user-defined item.
- an identifier starts with a letter A to Z or a to z or an underscore (_) followed by zero or more letters, underscores, and digits (0 to 9).
- C++ does not allow punctuation characters such as @, $, and % within identifiers.
- C++ is a case-sensitive programming language.
  - Example:
    - int a; // correct
    - int _0bbc; //correct
    - float @abc; //in correct

C++ Keywords

- whose meaning is already defined to a compiler.
- Can't use them to refer a variable name.
- They are reserved by C++ for specific purposes and cannot be used as identifiers.

E.g.

asm	**continue**	**float**	**new**	**signed**	**try**
auto	**default**	**for**	**operator**	**sizeof**	**typedef**
break	**delete**	**friend**	**private**	**static**	**union**
case	**do**	**goto**	**protected**	**struct**	**unsigned**
catch	**double**	**if**	**public**	**switch**	**virtual**
char	**else**	**inline**	**register**	**template**	**void**
class	**Enum**	**int**	**return**	**this**	**volatile**
const	**extern**	**long**	**short**	**throw**	**while**

Data types

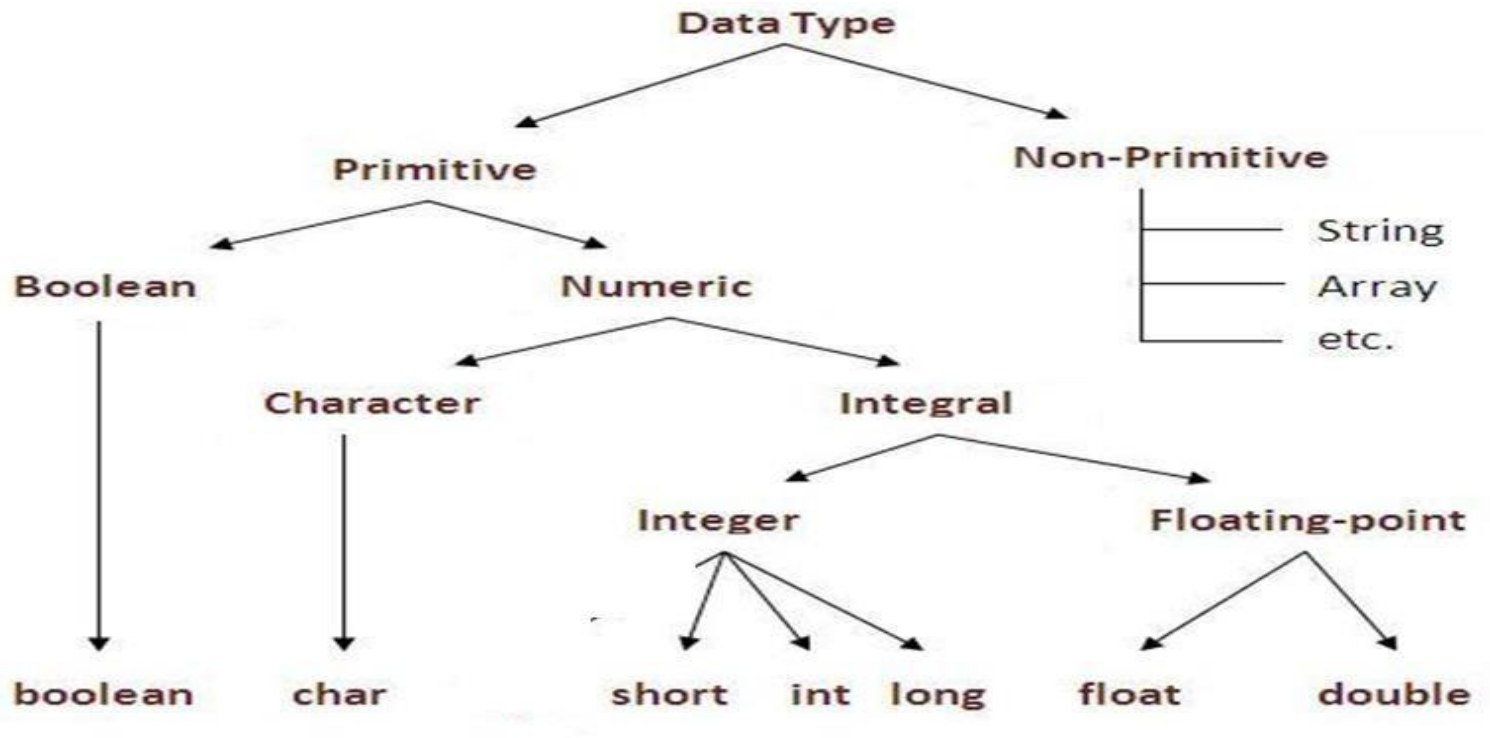
- A data type specifies the type of data that a variable can store such as integer, floating, character etc.
- There are four types of data types in C++:-
  - Basic {primitive Data Type}
  - Derived
  - Enumeration
  - User defined

Basic	int, char, float, double, bool etc
Derived	array, pointer, union, class etc
Enumeration	enum
User defined	structure

Cont'd ...

There are two data types

- Primitive Data Types
- Reference/Object Data Types



Cont'd ...

- There are seven primitive data types supported by C++ programming.

- Primitive data types are predefined by the language and named by a keyword.
  - Short
  - int
  - Long
  - float
  - Double
  - Bool
  - char

Variable

- A variable tells the compiler where and how much storage to create for the variable.
- Variables are nothing but reserved memory locations to store values.
- Variables are places where information can be stored while a program is running.
- It is used to store data. Its value can be changed and it can be reused many times.

Example:-

```
int x;
float y;
char z;
```

```
int x=5, b=10;
float f=30.8;
char c='A';
```

C++ Literals

- A literal is a source code representation of a fixed value.
- They are represented directly in the code without any computation.
- C++ supports few special literals as integers, float, and characters.

\n	Newline (0x0a)
\f	Formfeed (0x0c)
\b	Backspace (0x08)
\s	Space (0x20)
\t	tab
\"	Double quote
\'	Single quote
\\	Backslash
\?	Question mark

C++ Operators

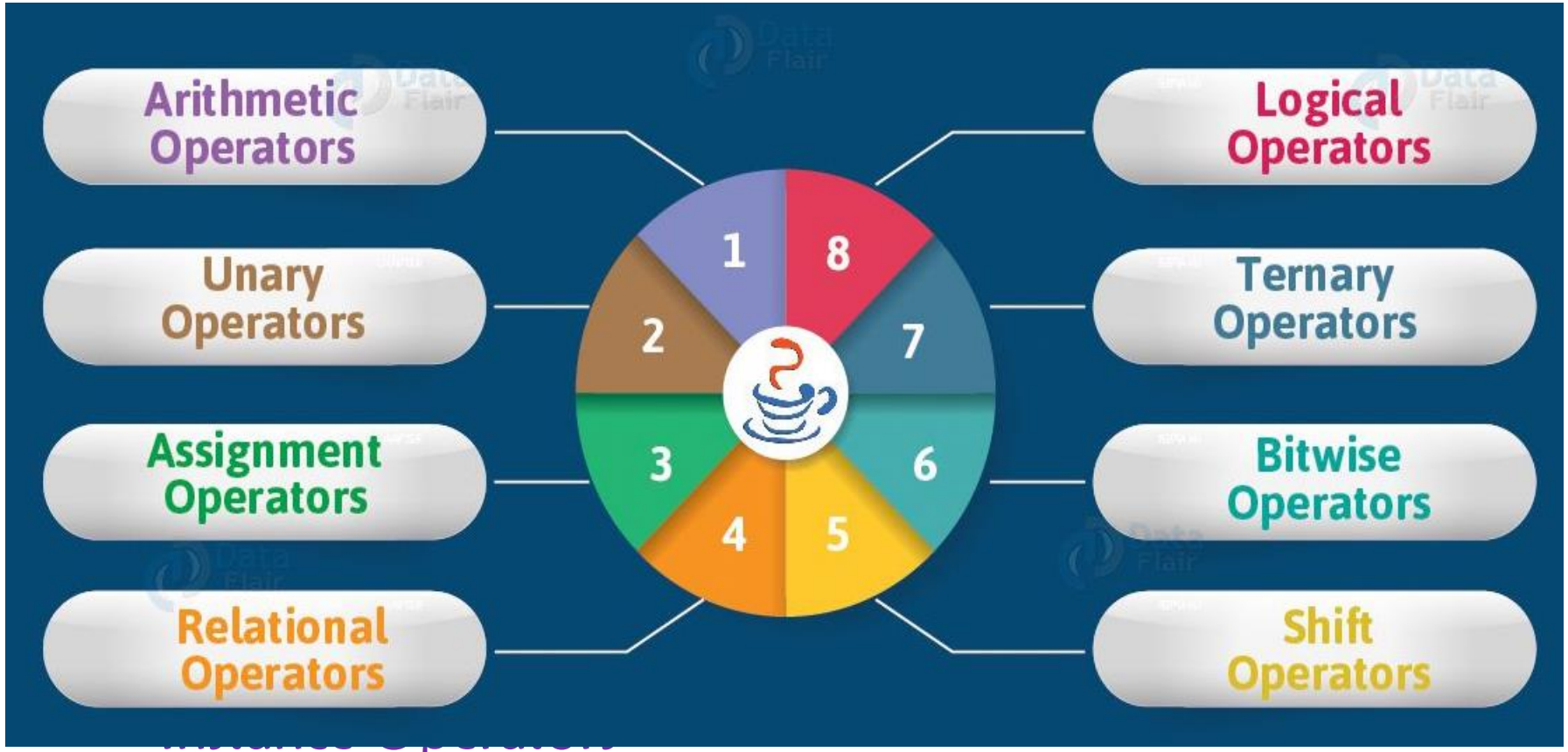
- Operators are special symbols used for
  - mathematical functions
  - assignment statements
  - logical comparisons
  - Manipulate variables
- An operator is a symbol that operates one or more arguments to produce a result.
- Examples:

```
3 + 5   // uses + operator
14 + 5 – 4 * (5 – 3)    // uses +, -, * operators
```

- Expressions can be combinations of variables, primitives and operators that result in a value


Types of Operator

- There are different types of operators:



Arithmetic Operators

- used in mathematical expressions in the same way that they are used in algebra.
- C++ has 6 basic arithmetic operators

+	addition
-	subtraction
*	multiplication
/	division
%	modulo (remainder)
^	exponent (to the power of)

- Order of operations (or precedence) when evaluating an expression is the same as you learned in school(Algebra).

Assignment Operator

- The basic assignment operator (=) assigns the value of var to expr

  var = expr ;

- C++ allows you to combine arithmetic and assignment operators into a single operator.

- Examples:

  x = x + 5; is equivalent to x += 5;

  y = y * 7;is equivalent to y *= 7;

Increment/Decrement Operators

Increment Operator:-

count = count + 1;

can be written as:

++count; or count++;

++ is called the increment operator.

Decrement Operator:

count = count - 1;

can be written as:

--count; or count--;

-- is called the decrement operator.

Relational (Comparison) Operators

- Relational operators compare two values
- Produces a boolean value (true or false) depending on the relationship

operation	is true when . . .
a > b	a is greater than b
a >= b	a is greater than or equal to b
a == b	a is equal to b
a != b	a is not equal to b
a <= b	a is less than or equal to b
a < b	a is less than b

Logical Operators

- Logical operators can be referred to as Boolean operators.
  - because they are only used to combine expressions that have a value of true or false.
  - Assume Boolean variables A holds true and variable B holds false, then

Operator	Description	Example
&&	Logical AND operator	(A && B) is false
||	OR Operator.	(A || B) is true
!	NOT Operator	!(A && B) is true

QUIZ

1) What is the value of number?

```
int number = 5 * 3 – 3 / 6 – 9 * 3;
```

-12

2) What is the value of result?

```
int x = 8;
int y = 2;
bool result = (15 == x * y);
```

false

3) What is the value of result?

```
bool x = 7;
bool result = (x < 8) && (x > 4);
```

true

4) What is the value of numCars?

```
int numBlueCars = 5;
int numGreenCars = 10;
int numCars = numGreenCars++ + numBlueCars +
++numGreeenCars;
```

27

Conditional Operators

- Conditional operator is also known as the ternary operator (?).
- It consists of three operands and is used to evaluate Boolean expressions.
- The goal of the operator is to decide, which value should be assigned to the variable.
- Syntax
  - variable x = (expression) ? value if true : value if false

Bitwise Operators

- works on bits and performs bit-by-bit operation.
- There are four basic bitwise operators in C++. These are:
  - & (bitwise and
  - | (bitwise or)
  - ^ (bitwise XOR)
  - ~ (bitwise compliment)

Example:-

- Assume integer variable A holds 60 and variable B holds 13 then

Operator	Description	Example
& (bitwise and)	Binary AND Operator copies a bit to the result if it exists in both operands.	(A & B) will give 12 which is 0000 1100
| (bitwise or)	• Binary OR Operator copies a bit if it exists in either operand.	(A | B) will give 61 which is 0011 1101
^ (bitwise XOR)	• Binary XOR Operator copies the bit if it is set in one operand but not both.	(A ^ B) will give 49 which is 0011 0001
~ (bitwise compliment)	• unary and has the effect of 'flipping' bits.	(~A) will give -61 which is 1100 0011 in 2's complement form

Bitwise Operators

Example:

- Assume if a = 60 and b = 13;

- now in binary format they will be as follows –

a = 0011 1100

b = 0000 1101

---

a&b = 0000 1100

a|b = 0011 1101

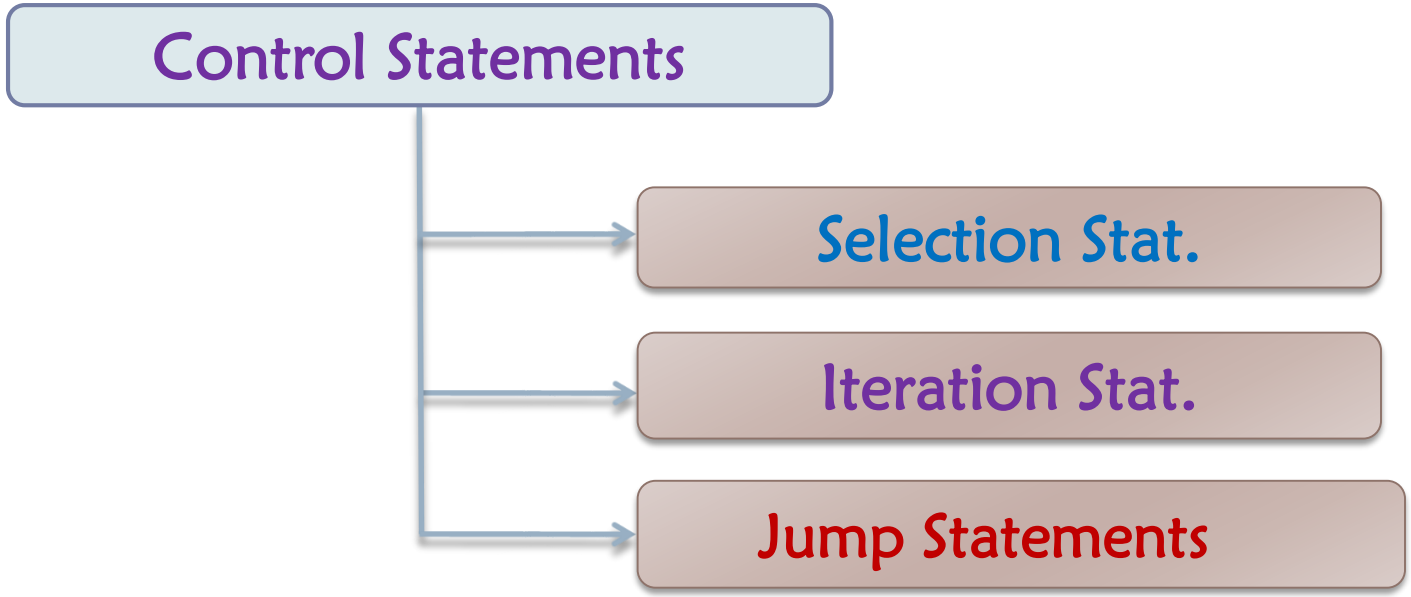
a^b = 0011 0001

~a = 1100 0011

Control Statements

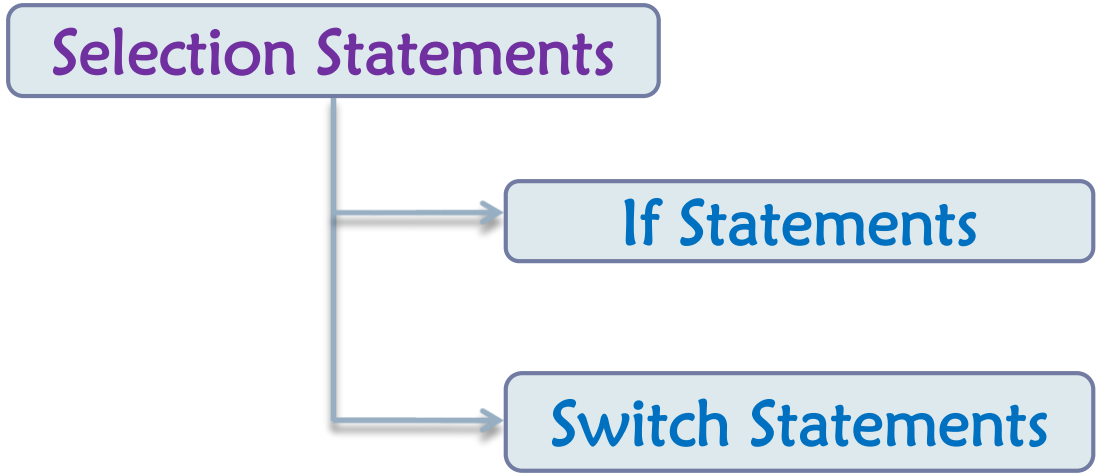
- Control statements alter the flow of the program
- Used to cause the flow of control to advance and branch based on changes to the state of a program.

- C++ control statements are categorized in to three.



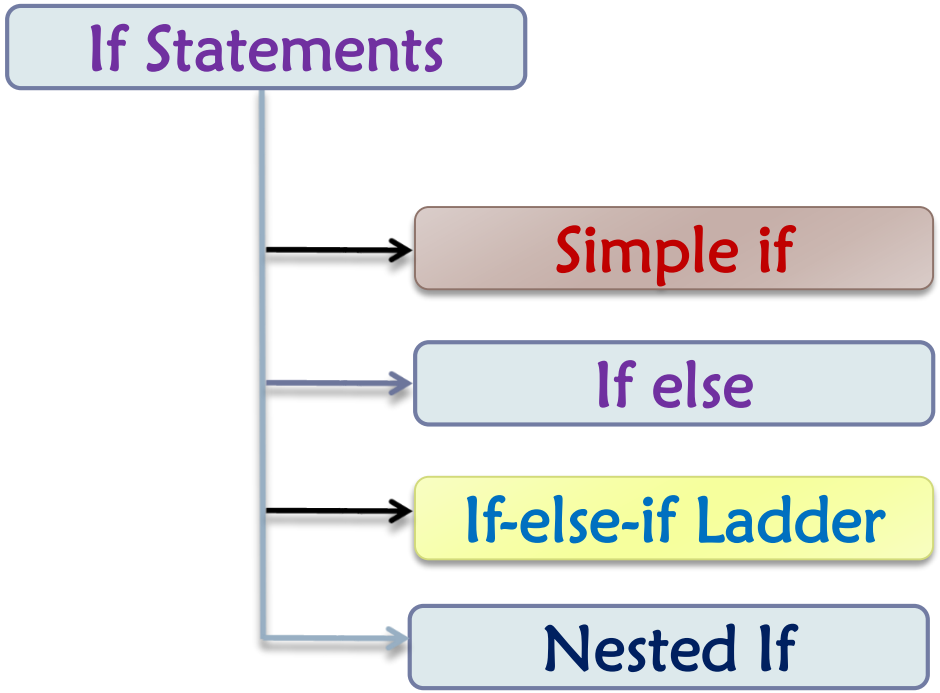
Selection Statements

- It is also called as a decision making statements.
- used to choose different paths of execution based upon the outcome of an expression or the state of a variable.
- There are two types of selection/decisions statements in C++:



If Statements

- If statement is used to test the condition.
- It checks Boolean condition: true or false.
- There are four types of if statements in C++. These are:-



Simple If Statement

- The statements will be evaluated if the value of the condition is true.

Syntax:

```
if (Condition)
    {
  statement1;
    }
rest_of_program
```

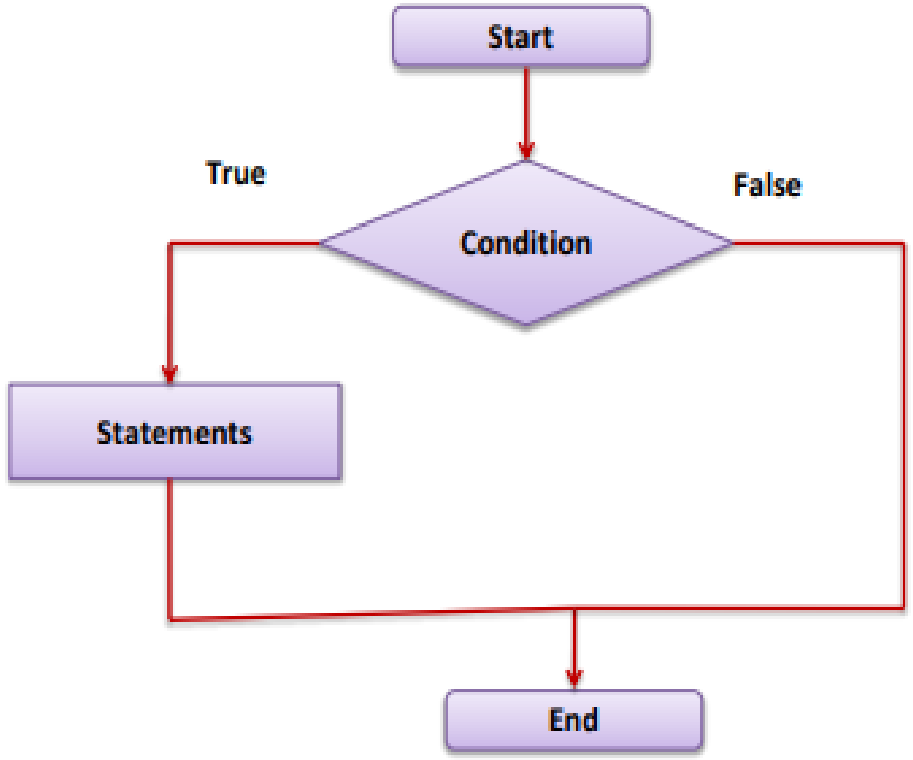
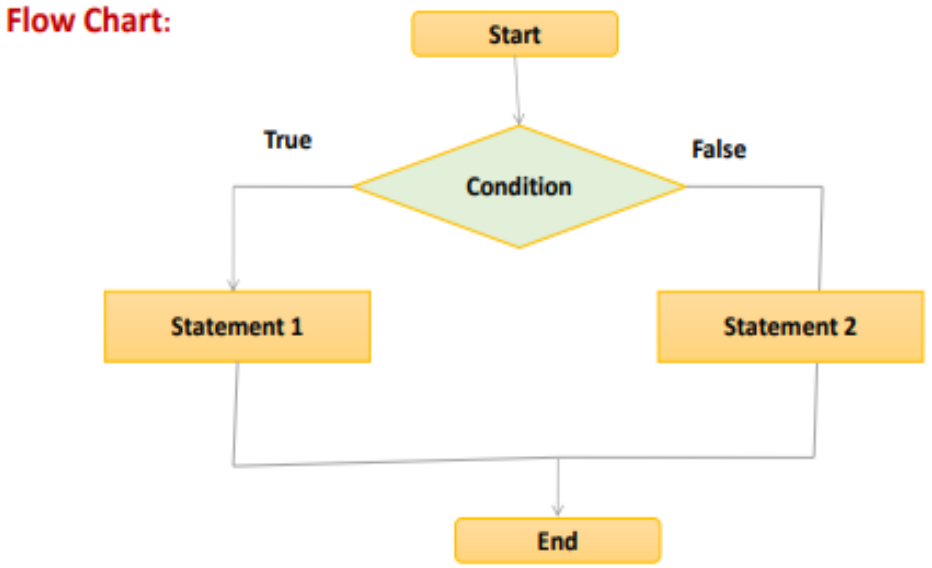


Fig. 1 Simple If Statement Flow chart

If-Else Statement

- If-else followed by an optional else statement, which executes when the Boolean expression is false.

```
if (Condition)
   {
  statement1;
    }
   else
    {
  statement2;
    }
 next_statement;
```



- Statement 1 is evaluated if the value of the condition is true otherwise statement 2 is evaluated.
  - It is used to take decision based on a single condition

If-else-if ladder Statement

- Executes one condition from multiple statements.
- very useful to test various conditions using single if...else if statement.
- Used when:
  - An if can have zero or one else's and it must come after any else if's.
  - An if can have zero to many else if's and they must come before the else.
  - Once an else if succeeds, none of the remaining else if's or else's will be tested.

Cont'd …

- Syntax:

```
if(condition1){
Statement 1 ;
}
else if(condition2){
Statement 2 ;
}
else if(condition3){
Statement 3;
}
...
else{
Statement n;
}
```

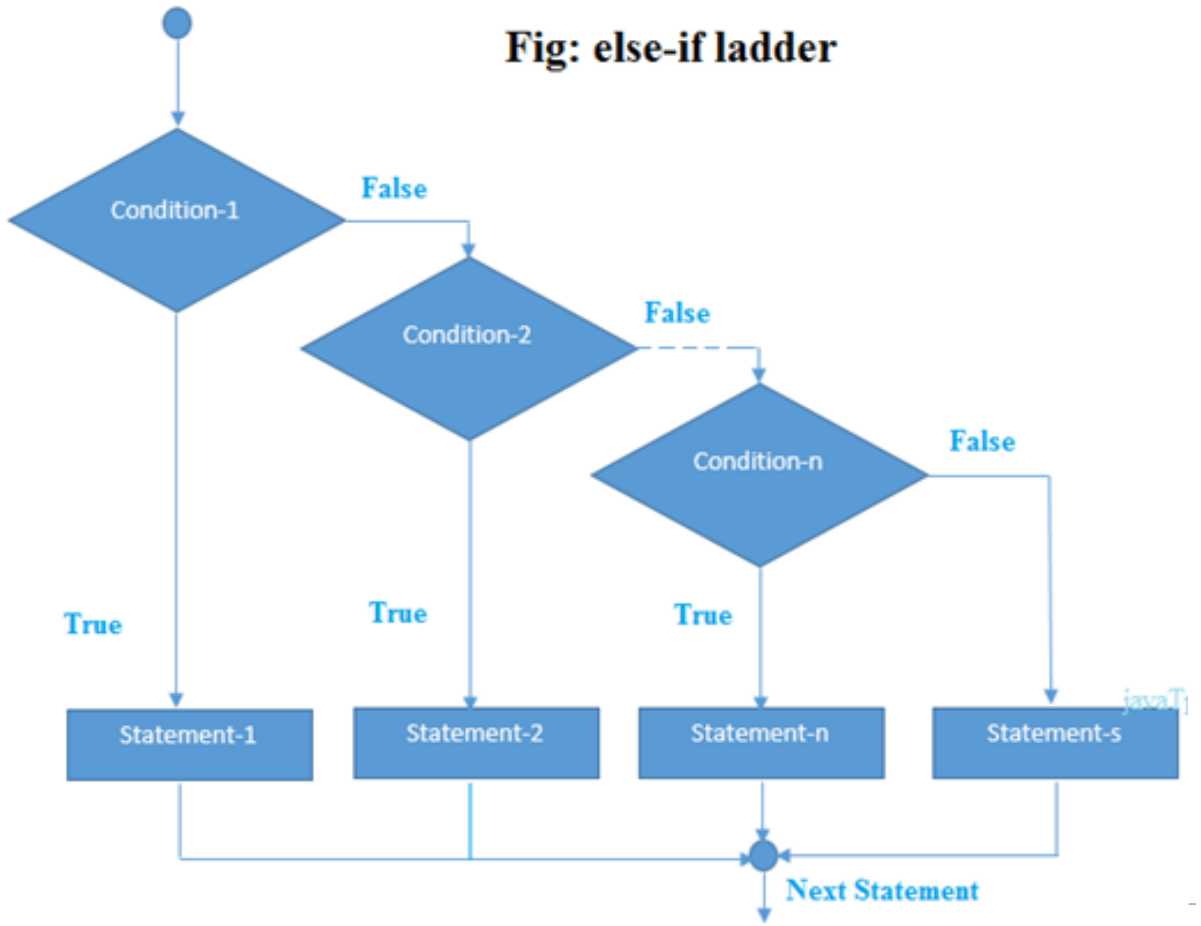


Fig: else-if ladder

nested if statement

- It is refer to the else if statement inside another if or else if statement.
- A nested if is an if statement that is the target of another if or else.
- Nested ifs are very common in programming.

Syntax:

```
if(Boolean_expression 1) {
        // Executes when the Boolean expression 1 is true

    if(Boolean_expression 2) {
        // Executes when the Boolean expression 2 is true
    }
}
```

Example1:

```
public class Test1 {
   public static void main(String args[]) {
         int x = 30;
          int y = 10;
      if( x == 30 ) {
         if( y == 10 ) {
            System.out.print(“ X = 30 and Y = 10");
         }
      }
   }
}
```

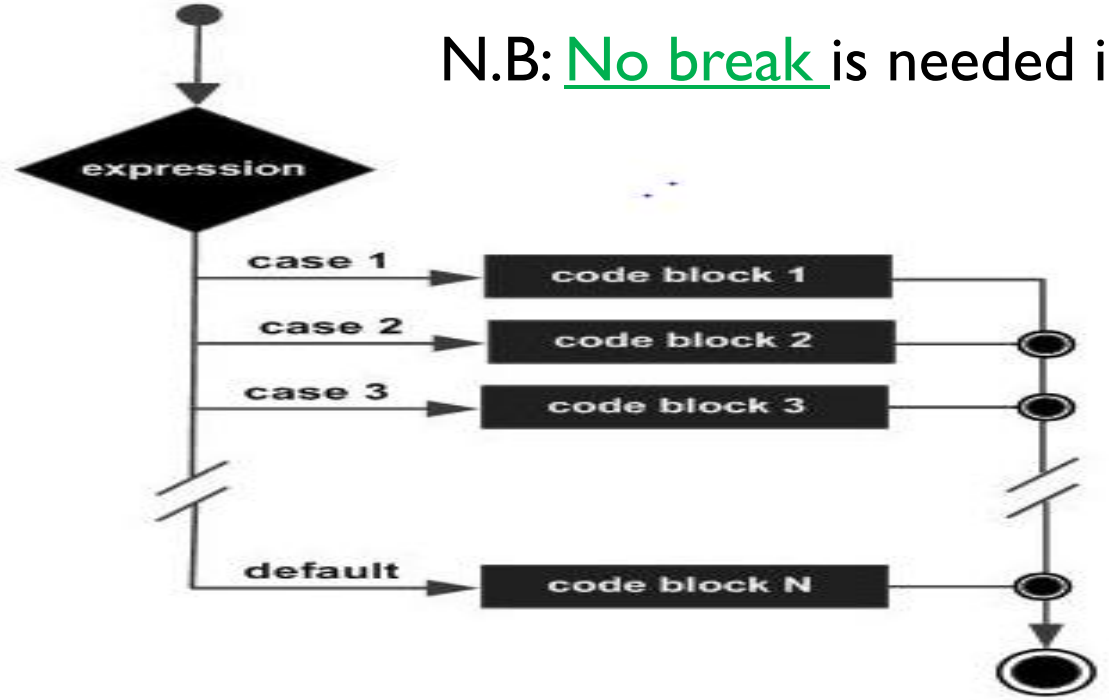
Switch Statements

- executes one statement from multiple conditions [as if else if]
- allows a variable to be tested for equality against a list of values.
- Each value is called a case, and the variable being switched on is checked for each case.
- You can have any number of case statements.
  - Syntax:

```
switch(expression) {
case value 1:
      // statements;
      break;
case value 2:
   // Statements
    break;
default : // Optional    }
```

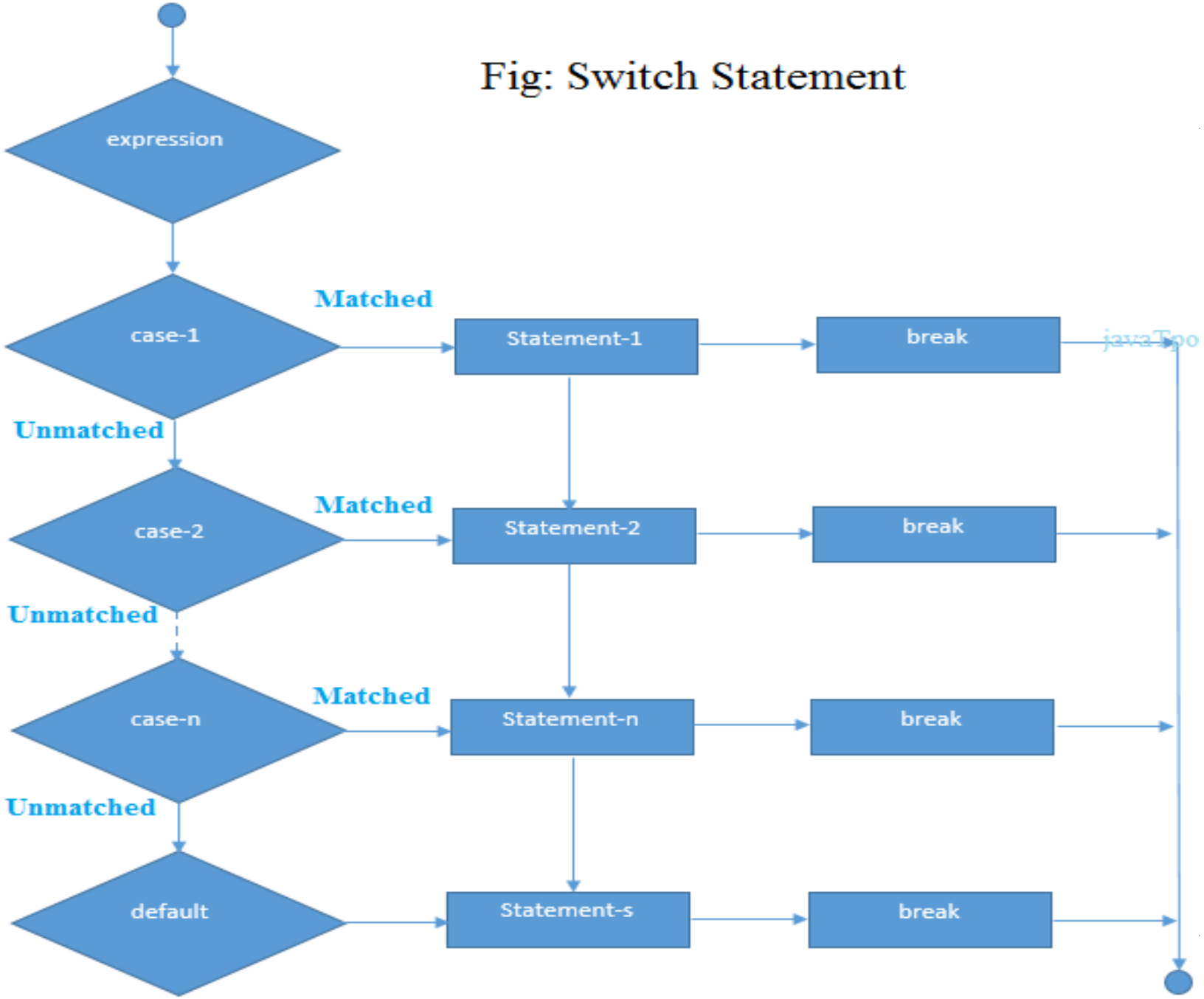
Cont'd …

- When a break statement is reached, the switch terminates, and the flow of control jumps to the next line following the switch statement.
- If no break appears, the flow of control will fall through to subsequent cases until a break is reached.



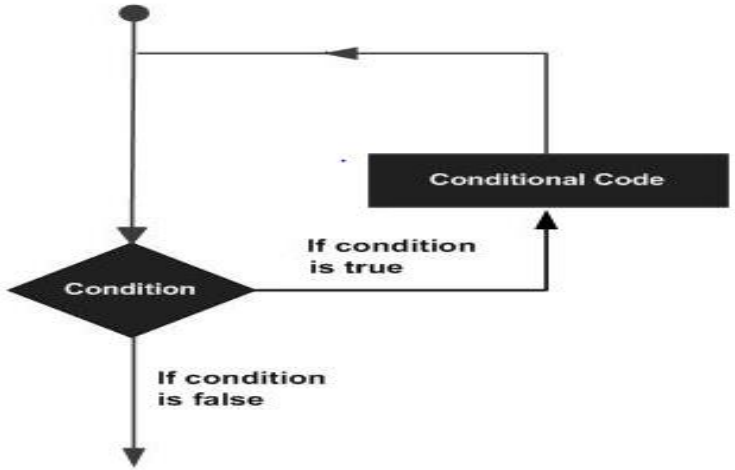
N.B: No break is needed in the default case

Fig: Switch Statement

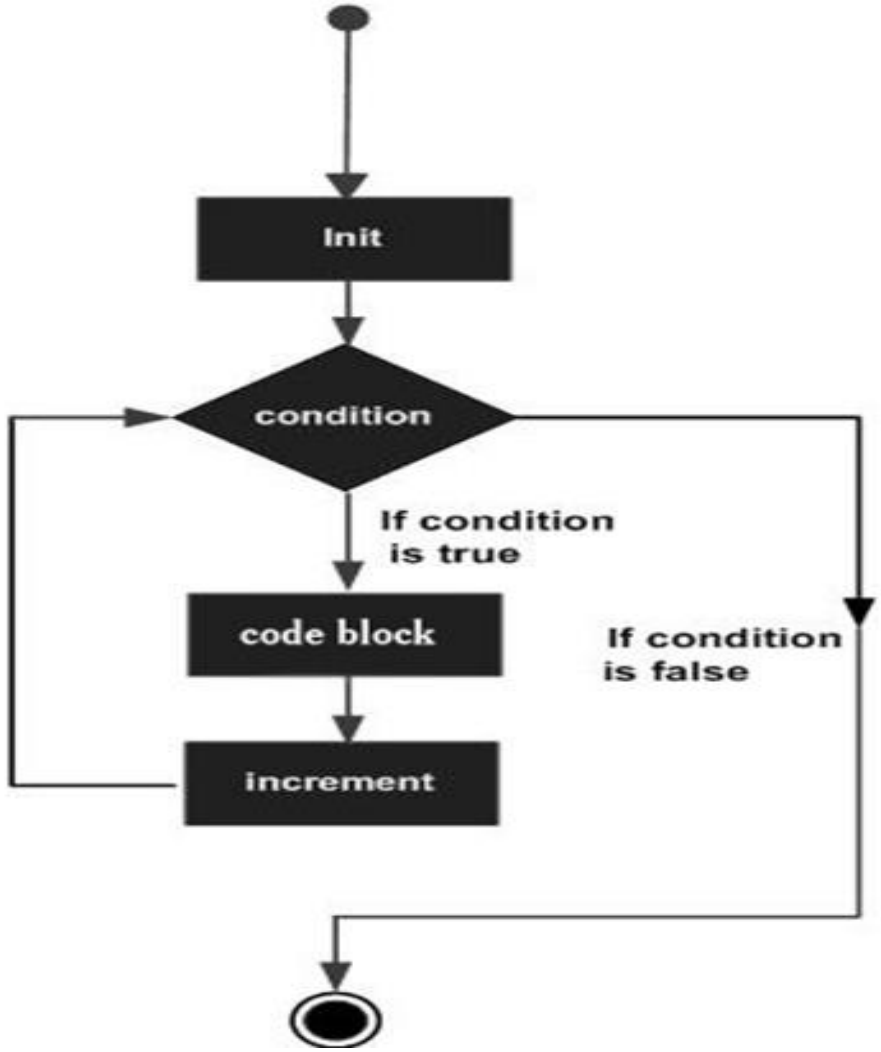


Iteration Statements

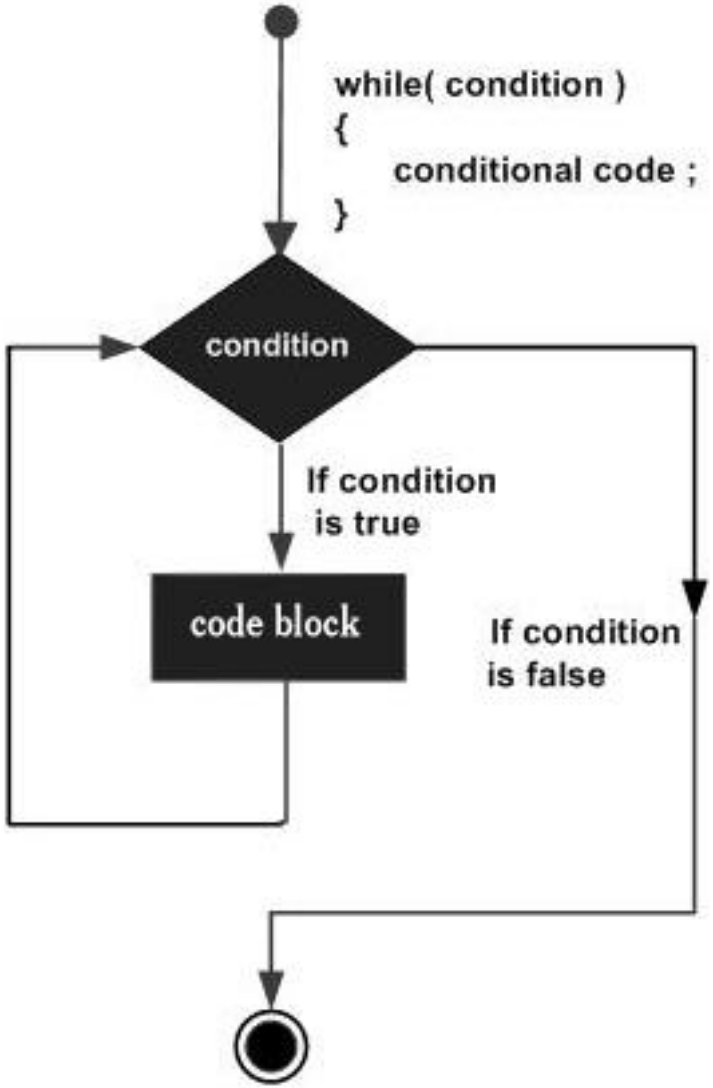
- Also known as a looping statements.
- It allows to you to execute a statement or block of statements repeatedly.
- Executes a block of statements when a particular condition is true
- There are three types of loops in C++:
  - for loops
  - while loops
  - do-while loops



Cont'd …


Init
condition
If condition
is true
code block
If condition
is false
increment

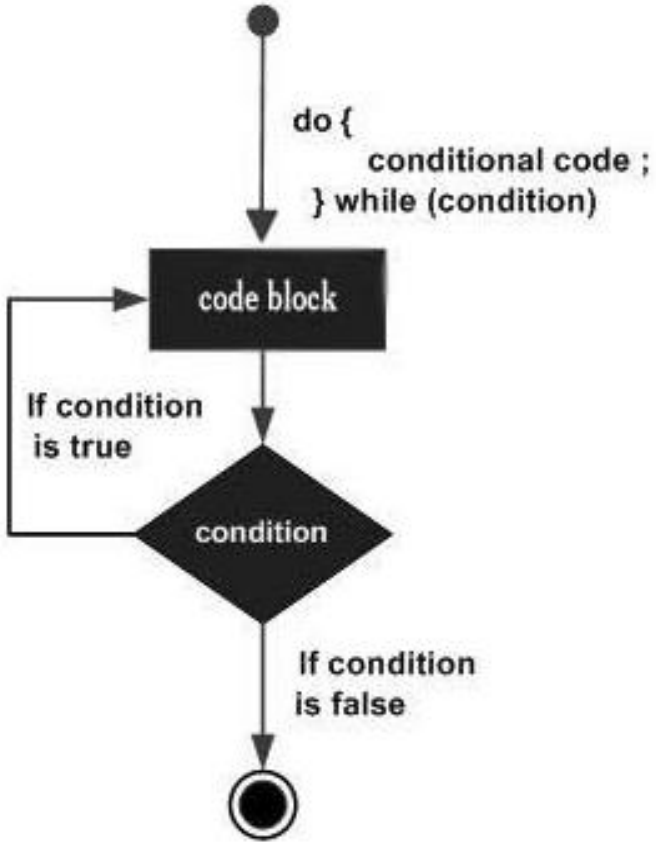
Cont'd …



1 + 2 + 3 + 4 + 5 + 6 + 7 + 8
+ 9 + 10 = 55

do … while

- It is similar to a while loop, except that a do...while loop is guaranteed to execute at least one time.

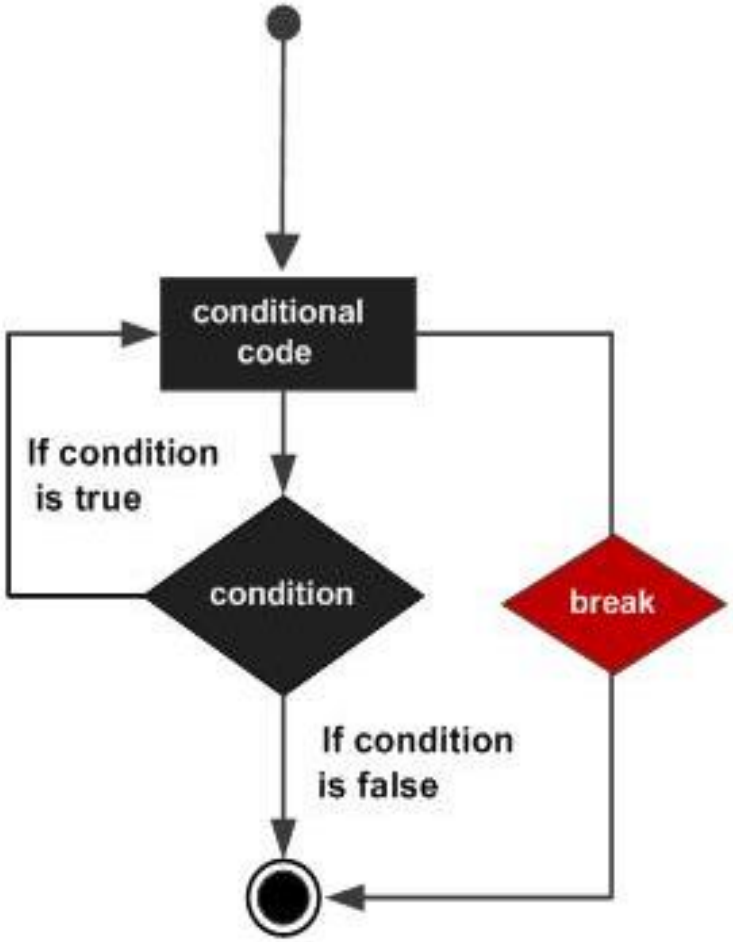


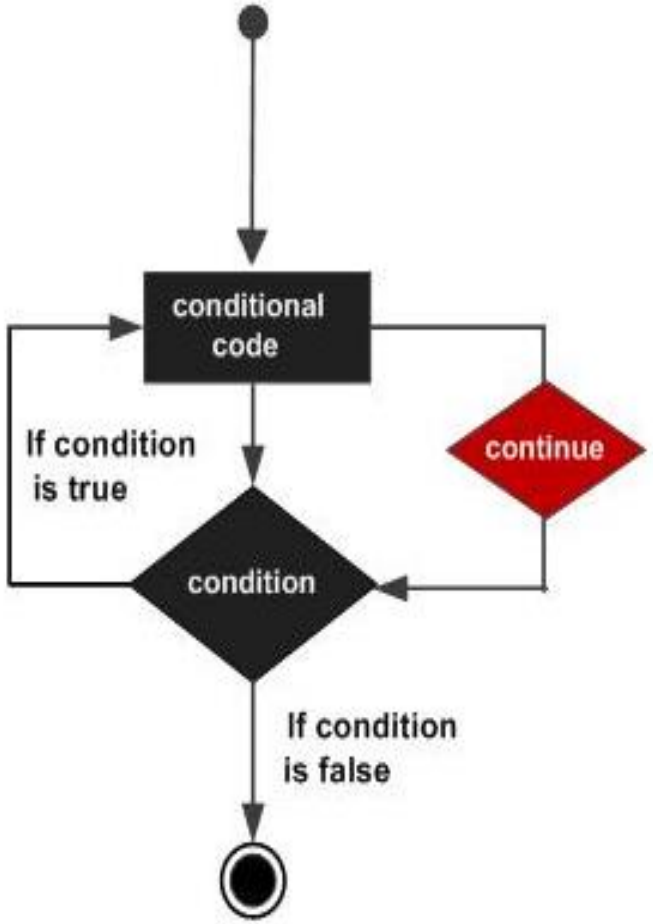

```
Syntax:
        initialization;
        do {
          // Statements;
          // increment/decrement;
        }
        while(Boolean_expression);
```

Jump Statements

- Also known as a loop control statements
- Loop control statements change execution from its normal sequence.
- When execution leaves a scope, all automatic objects that were created in that scope are destroyed.

Cont'd ...


conditional code
If condition is true
condition
break
If condition is false


conditional code
If condition is true
continue
condition
If condition is false

Thank you!

Question

End of Part 1