Introduction to Basic Computer Programming MT 6302

What is C++?

- C++ is a cross-platform language that can be used to create high-performance applications.
- C++ was developed by Bjarne Stroustrup at Bell labs in 1979, as an extension to the C language.
- C++ gives programmers a high level of control over system resources and memory.

Why Use C++

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

C++ Syntax

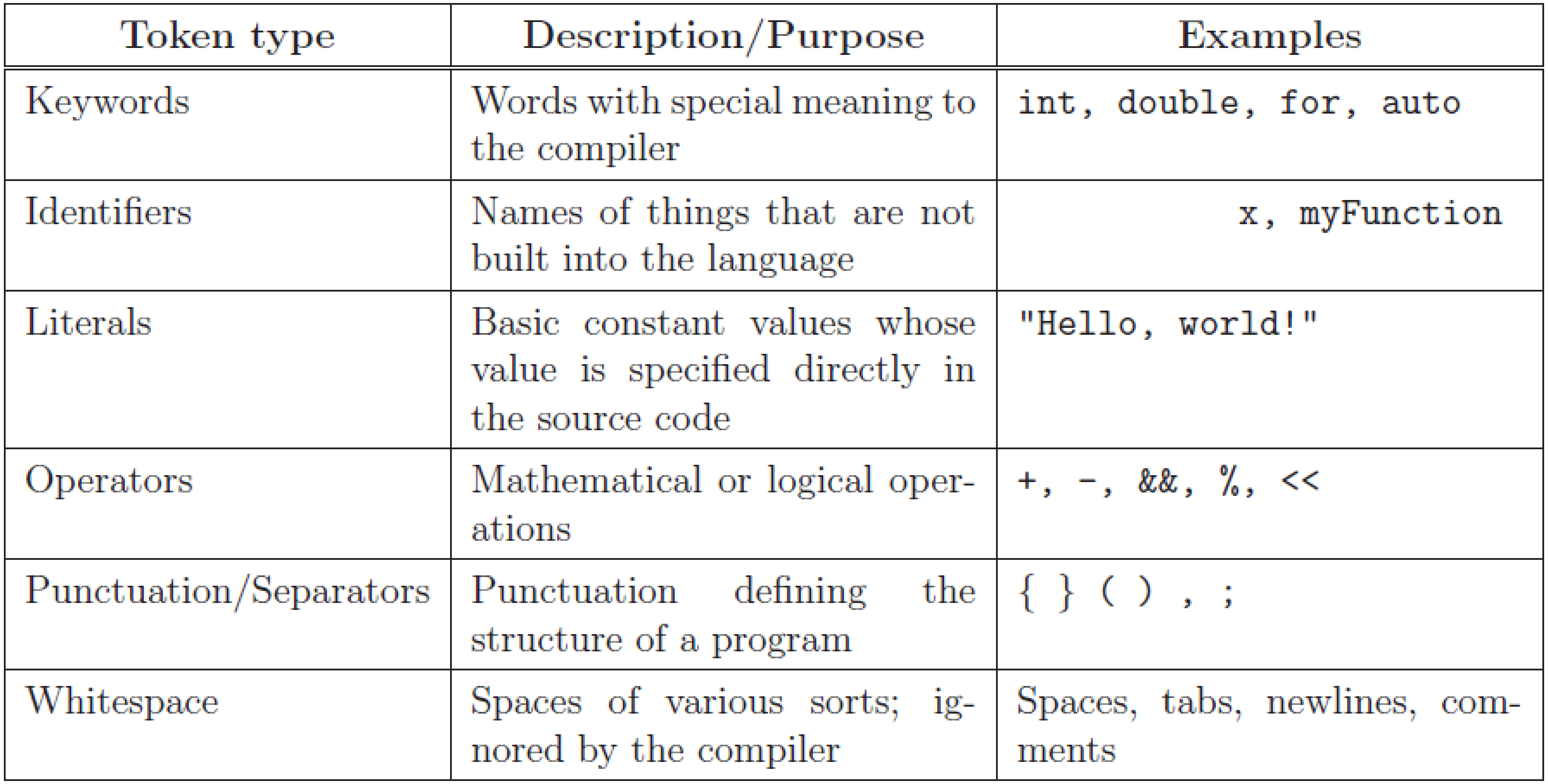
```
#include <iostream>
using namespace std;

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

- **Line 1:** #include <iostream> is a **header file library** that lets us work with input and output objects, such as cout (used in line 5). Header files add functionality to C++ programs.
- **Line 2: using namespace std** means that we can use names for objects and variables from the standard library.
- **Line 3:** A blank line. C++ ignores white space.
- **Line 4:** Another thing that always appear in a C++ program, is **int main().** This is called a **function**. Any code inside its **curly brackets {}** will be executed.
- **Line 5:** cout (pronounced "see-out") is an **object** used to output/print text. In our example it will output "Hello World".
- **Line 6:** return 0 ends the main function.

Tokens

Tokens are the minimal chunk of program that have meaning to the compiler or the smallest meaningful symbols in the language.



Token type	Description/Purpose	Examples
Keywords	Words with special meaning to the compiler	`int, double, for, auto`
Identifiers	Names of things that are not built into the language	`x, myFunction`
Literals	Basic constant values whose value is specified directly in the source code	`"Hello, world!"`
Operators	Mathematical or logical operations	`+, -, &&, %, <<`
Punctuation/Separators	Punctuation defining the structure of a program	`{ } ( ) , ;`
Whitespace	Spaces of various sorts; ignored by the compiler	Spaces, tabs, newlines, comments

Identifiers

Identifiers are user defined word used to name of entities like variables, arrays, functions, structures etc.

Example :

- STDNAME
- *SUB*
- TOT_MARKS
- _TEMP

Rules for naming identifiers are:

i. Name should only consist of alphabets (both upper and lower case), digits and underscore (_) sign.

ii. First characters should be alphabet or underscore

iii. Name should not be a keyword

iv. Since C++ is a case sensitive, the upper case and lower case considered differently, for example code, Code, CODE etc. are different identifiers.

Keywords

- ❖ **Keywords** words reserved for doing specific task, These words are predefined and always written in lower case or small letter in C++.
- ❖ These keywords can not be used as a variable name as it assigned with fixed meaning.

Some examples are int, short, signed,

unsigned, default, volatile, float, long, double, break, continue, typedef, static,do, for, union, return, while, do, extern, register, enum, case, goto, struct,char, auto, const etc.

C++ Variables

- ❖Variables are defined as reserved memory space which stores a value of a definite datatype.
- ❖Rules for naming the variables are the same as the naming identifiers. Before used in the program it must be declared. Declaration of variables specify its name, data types and range of the value that variables can store depends upon its data types.

Declaration of Variables

❖A variable can be used to store a value of any data type. The declaration of variables must be done before they are used in the program. The general format for declaring a variable.

Syntax : data_typevariable-1;

❖Variables are separated by commas and declaration statement ends with a semicolon.

Ex : int x,y; float a; char m;

Assigning values to variables (initialization)

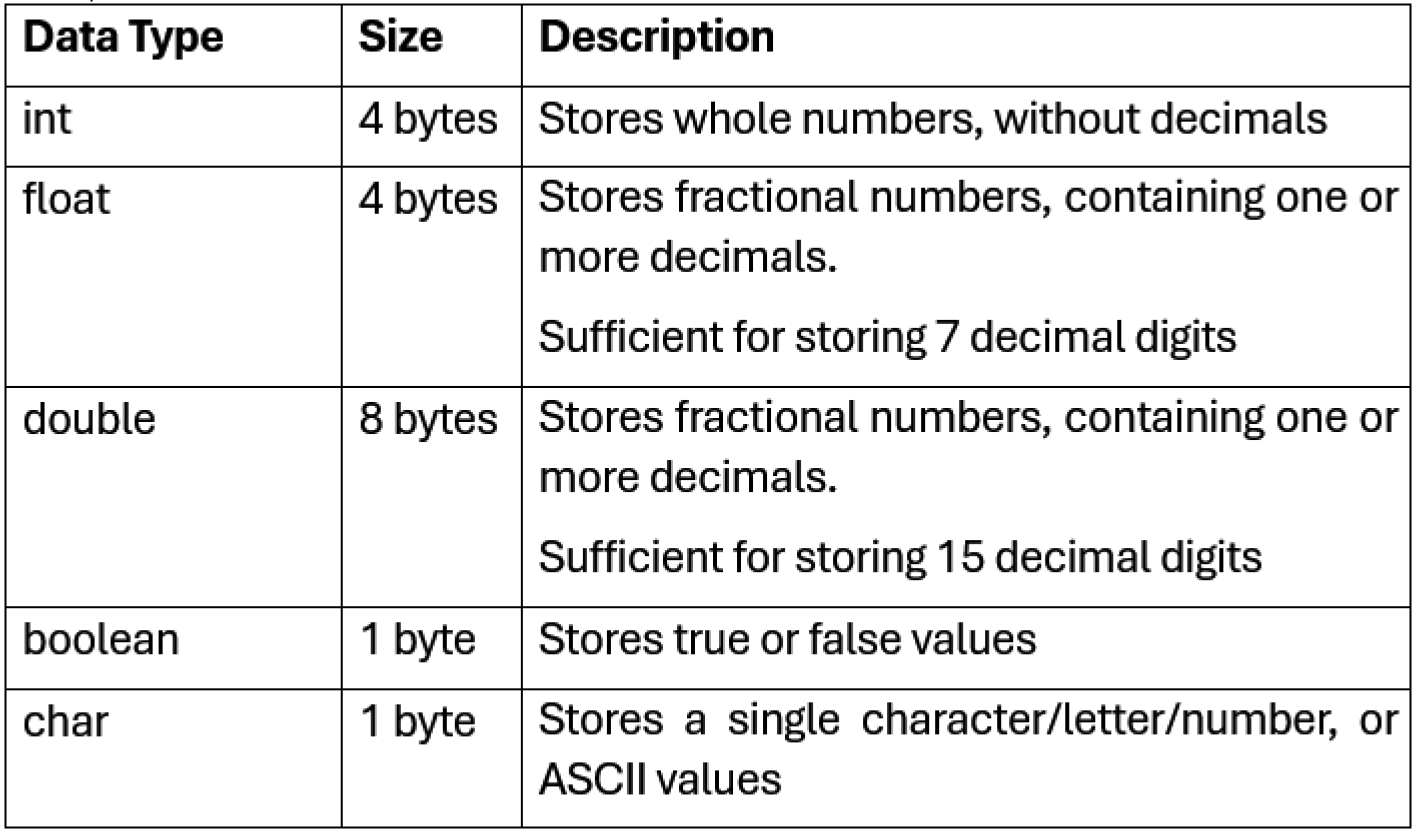
- ❖Values can be assigned to variables using the assignment operator (=). The general format statement is :

Syntax : variable = value;

Ex : x=100;   a= 12.25;

Basic Data Types

- ❖The data type specifies the size and type of information the variable will store



Data Type	Size	Description
int	4 bytes	Stores whole numbers, without decimals
float	4 bytes	Stores fractional numbers, containing one or more decimals. Sufficient for storing 7 decimal digits
double	8 bytes	Stores fractional numbers, containing one or more decimals. Sufficient for storing 15 decimal digits
boolean	1 byte	Stores true or false values
char	1 byte	Stores a single character/letter/number, or ASCII values

Strings

❖The string type is used to store a sequence of characters (text).

❖This is not a built-in type, but it behaves like one in its most basic usage. String values must be surrounded by double quotes:

Example:

```
string greeting = "Hello";
cout << greeting;
```

C++ Operators

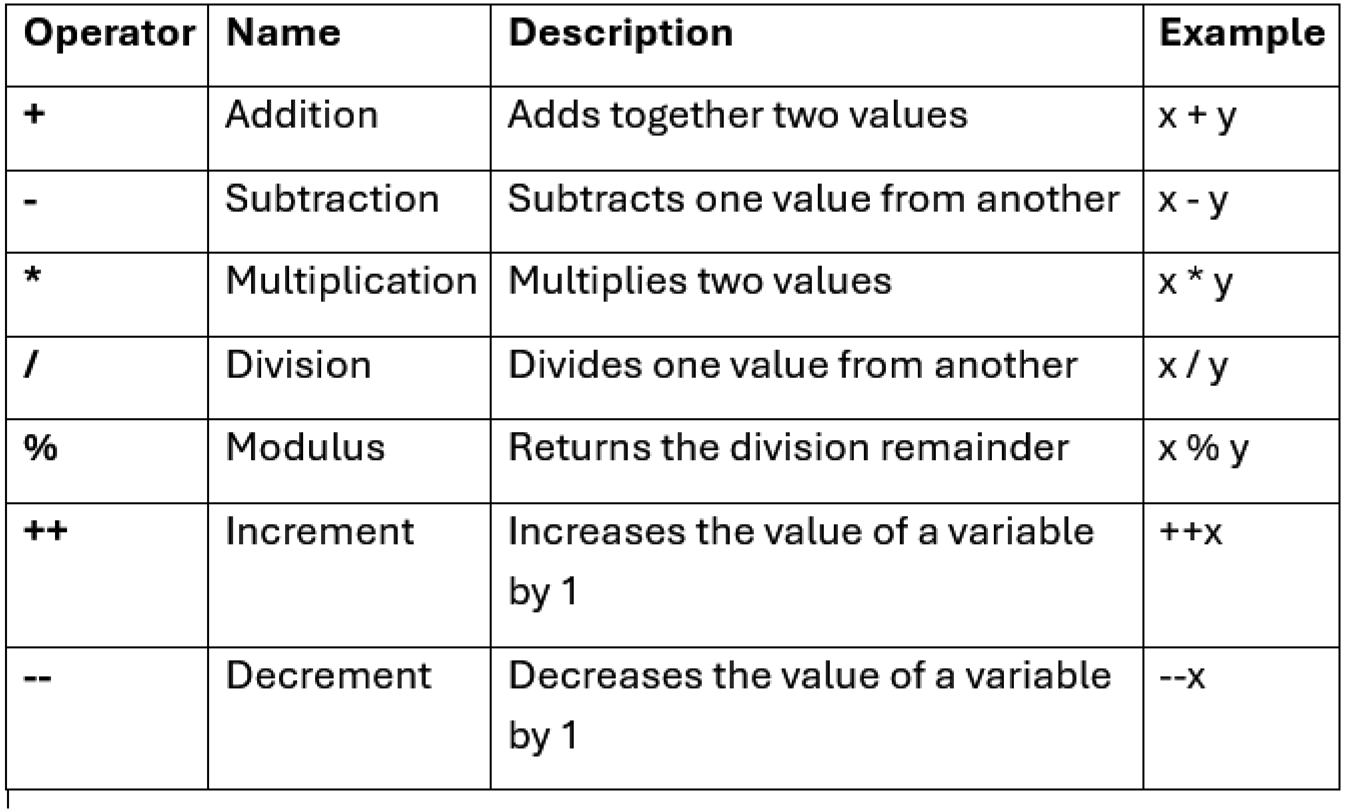
❖ Operators are used to perform operations on variables and values.

C++ divides the operators into the following groups:

i. Arithmetic operators
ii. Assignment operators
iii. Comparison operators
iv. Logical operators
v. Bitwise operators

Arithmetic Operators

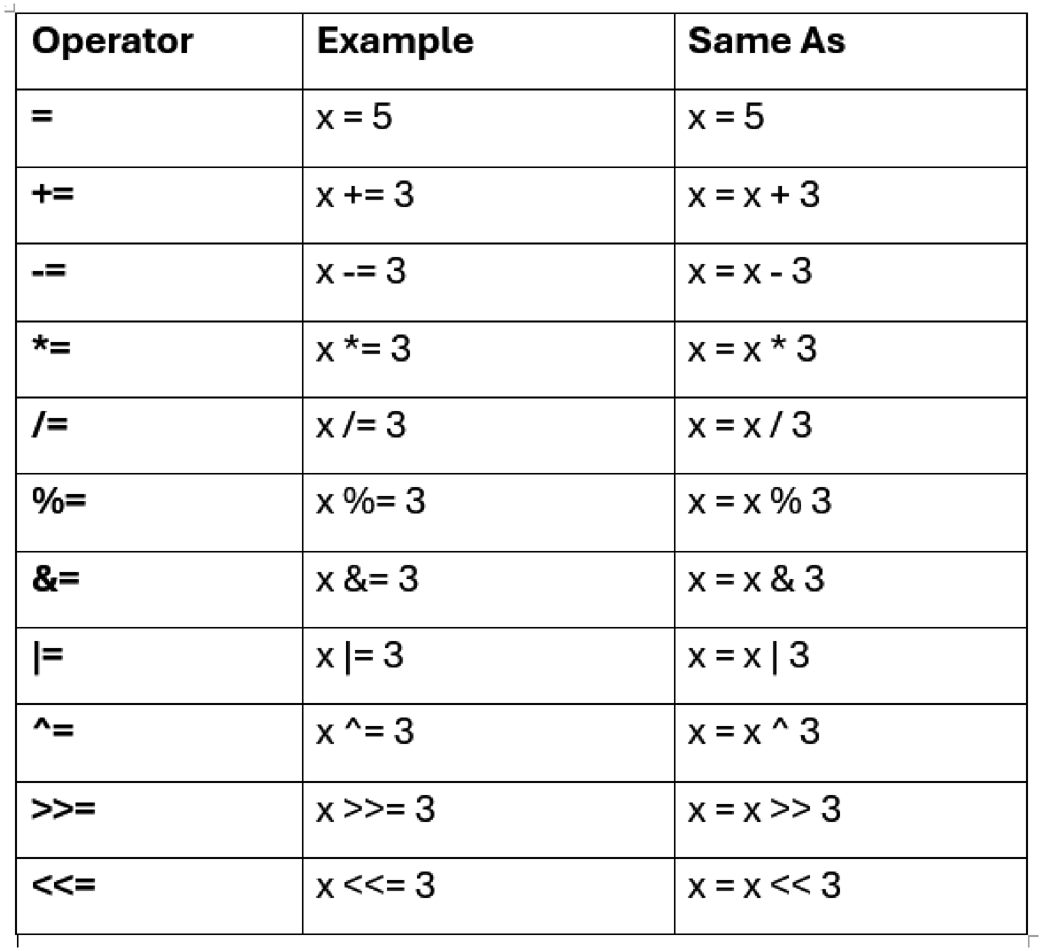
❖Arithmetic operators are used to perform common mathematical operations.



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 value from another	x / y
%	Modulus	Returns the division remainder	x % y
++	Increment	Increases the value of a variable by 1	++x
--	Decrement	Decreases the value of a variable by 1	--x

C++ Assignment Operators

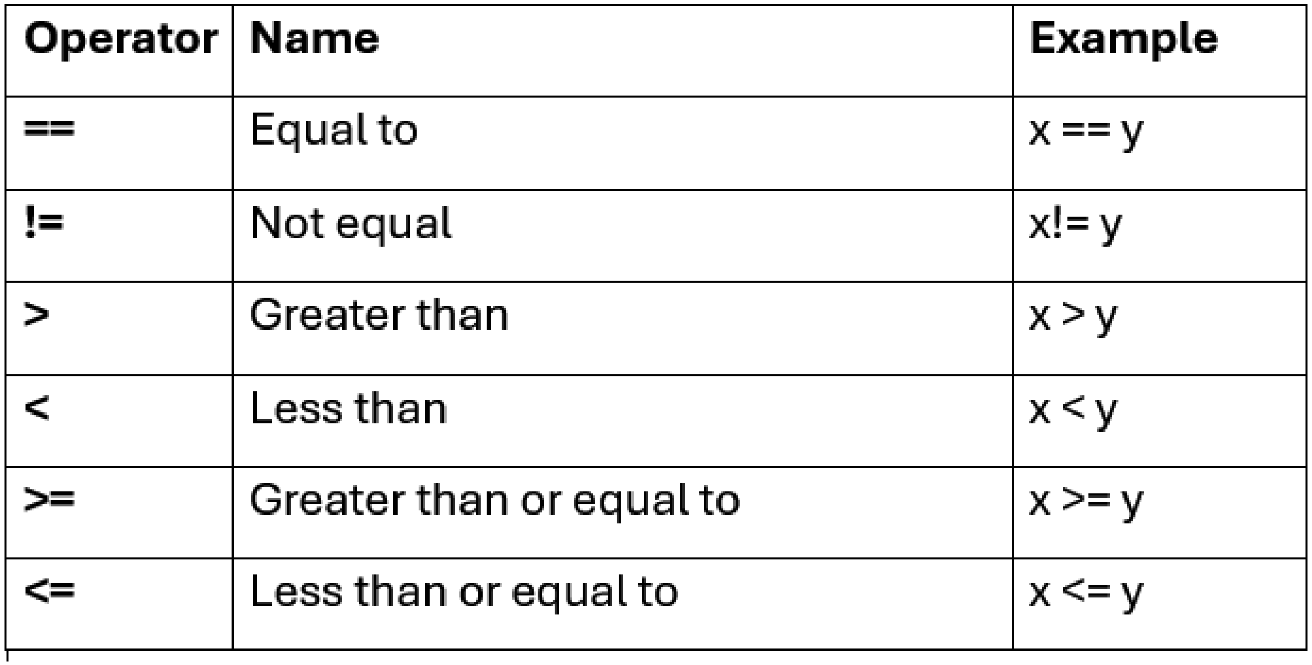
❖Assignment operators are used to assign values to variables.



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

C++ Comparison Operators

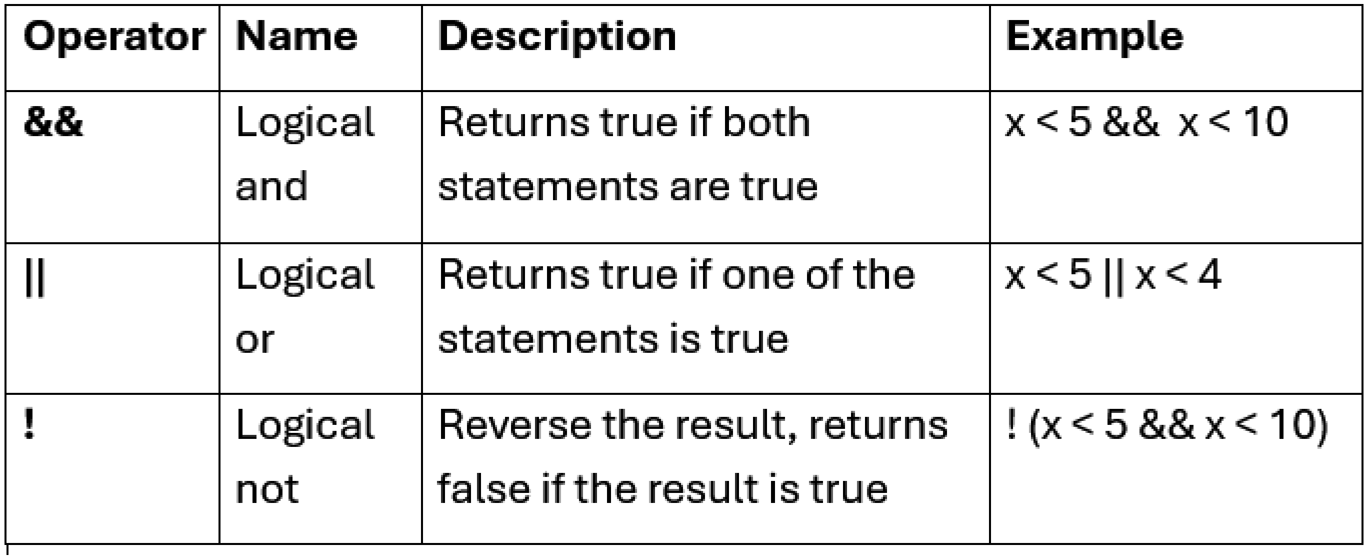
❖Comparison operators are used to compare two values.



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

C++ Logical Operators

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



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

Flow of Control

Control Structures

❖ *Control structures* are portions of program code that contain statements within them and, depending on the circumstances, execute these statements in a certain way.

❖ There are typically two kinds:

i. Conditionals and
ii. Loops.

Conditionals (Selection)

- ❖In order for a program to change its behavior depending on the input, there must a way to test that input. Conditionals allow the program to check the values of variables and to execute (or not execute) certain statements. C++ has if and switch-case conditional structures.

If statement

Syntax:

```
if(condition)
{
   statement1
   statement2…
}
```

If the condition resolves to a value of true, then the statements are executed before the program continues on. Otherwise, the statements are ignored.

if-else Statement

Syntax:

```
if(condition)
{
statementA1
statementA2
…
}
else
{
statementB1
statementB2
….
}
```

If the condition is met, the block corresponding to the `if` is executed. Otherwise, the block corresponding to the `else` is executed.

Because the condition is either satisfied or not, one of the blocks in an if-else *must* execute.

else if Statement

Syntax:

```
if(condition1)
{
     statementA1
     statementA2
     …
}
else if(condition2)
{
     statementB1
     statementB2
     …
}
else
```

else if Statement…

- If condition1 is met, the block corresponding to the if is executed. If not, then *only if* condition2 is met is the block corresponding to the else if executed. There may be more than one else if, each with its own condition. Once a block whose condition was met is executed, any else ifs after it are ignored. Therefore, in an *if-else-if* structure, either one or no block is executed.
- An else may be added to the end of an if-else-if. If none of the previous conditions are met, the else block is executed. In this structure, one of the blocks *must* execute, as in a normal if-else.

switch-case Statement

Syntax:

```
switch(expression)
{
case constant1:
statementA1
statementA2
...break;

case constant2:
statementB1
statementB2
...
break;
...
default:
statementZ1
statementZ2...
}
```

switch-case Statement…

- ❖The switch evaluates expression and, if expression is equal to constant1, then the statements beneath case constant 1: are executed until a break is encountered. If expression is not equal to constant1, then it is compared to constant2.
- ❖If these are equal, then the statements beneath case constant 2: are executed until a break is encountered.
- ❖If not, then the same process repeats for each of the constants, in turn. If none of the constants match, then the statements beneath default: are executed.

Loops

- ❖Conditionals execute certain statements if certain conditions are met;
- ❖Loops execute certain statements *while* certain conditions are met.
- ❖C++ has three kinds of loops:

i. while,

ii. do-while, and

iii. for.

while loop

Syntax:

```
while(condition)
{
statement1
statement2
…
}
```

As long as condition holds, the block of statements will be repeatedly executed.

do-while loop

The *do-while* loop is a variation that guarantees the block of statements will be executed *at least once*.

Syntax:

```
do
{
Statement1
Statement2
…
}
while(condition);
```

The block of statements is executed and then, if the condition holds, the program returns to the top of the block.

for loop

Syntax:

```
for(initialization; condition; incrementation)
{
statement1
statement2…
}
```

The for loop is designed to allow a counter variable that is initialized at the beginning of the loop and incremented (or decremented) on each iteration of the loop.

Nested Control Structures

- It is possible to place ifs inside of ifs and loops inside of loops by simply placing these structures inside the statement blocks.
- This allows for more complicated program behavior

Example:

```
1 #include <iostream>
2 using namespace std;
3
4 int main() {
5 int x = 6;
6 int y = 0;
7
8 if(x > y) {
9 cout << “x is greater than y\n”;
10 if(x == 6)
11 cout << “x is equal to 6\n”;
12 else1
3 cout << “x is not equalt to 6\n”;14 } else
15 cout << “x is not greater than y\n”;
16
17 return 0;
18 }
```

bye