MNNIT
COMPUTER CODING
CLUB

CLASS–1

Introduction to Programming

Installation And Setups

- For Windows
- https://www.youtube.com/watch?v=fsmVbLMzBoo&ab_channel=ProgrammingKnowledgeII
- For gcc compiler in Linux :
- https://www.youtube.com/watch?v=cotkJrewAz0&ab_channel=ProgrammingKnowledge2
- VS code installation :
- https://code.visualstudio.com/download

Programming Language: C++

- C++ was released in 1985 a modification to C which was developed by Dennis Ritche in 1972.
- It was not named C+ but C++, because of the increment operator.
- C++ introduces the concept of Object Oriented Programming, which is not present in C.
- MySQL, world's most popular database is written in C++.
- C++ is extensively used by NASA.
- C++ has a very rich library support. (what is a library?)
- Initially C++ was called 'The New C'.
- A large chunk of YouTube is written in C++.
- C++ is the core of Unity3D, used for making games.

Should I learn C or C++ ?

- C is a fundamental language, it is used in creating Operating Systems, embedded devices, kernels and drivers.
- C is the root language for C++, so why not first learn C rather than C++?
- C++ can be used in gaming, graphics, web browsers and with help of object oriented programming, it can be used in creating software as well.

REASONS TO USE C++ :

- More applications are written entirely in C++ rather than C, even operating systems that use C at their core are known for having C++ for boosting complexity and functionality.
- One of the biggest reasons to use C++ is the library functions that help us to focous on logic rather than remembering code.

```c
int MAXSIZE = 8;
int stack[8];
int top = -1;

int isempty() {

   if(top == -1)
      return 1;
   else
      return 0;
}

int isfull() {

   if(top == MAXSIZE)
      return 1;
   else
      return 0;
}

int peek() {
   return stack[top];
}

int pop() {
   int data;

   if(!isempty()) {
      data = stack[top];
      top = top - 1;
      return data;
   } else {
      printf("Could not retrieve data, Stack is empty.\n");
   }
}

int push(int data) {

   if(!isfull()) {
      top = top + 1;
      stack[top] = data;
   } else {
      printf("Could not insert data, Stack is full.\n");
   }
}
```

```cpp
#include <iostream>
#include <stack>
using namespace std;
int main() {
    stack<int> stack;
}
```

C code for stack vs C++ code for stack

Hello World Program

Step 1: Type this code in a code editor. You can use Sublime Text , notepad, VS Code or any other editor of your choice.

Step 2: Create a folder named practice in desktop.

Step 3: Save the file with name helloworld.cpp in this folder.

Step 4: Type command

- g++ helloworld.cpp
- Check for any error

Step 5: To run the program type command

- .\a.exe (for windows)
- ./a.out (for linux)

```cpp
#include <iostream>
using namespace std;
int main()
{
    cout << "Hello World";
    return 0;
}
```

Output:

```
Hello World
```

Points To Notice about structure of a C++ program:

- Preprocessor directives (#)
- Header File
- namespace std
- main Function
- return at the end of program.

Components of A C++ Program

#include <iostream>	#include is **a preprocessor directive that tells the preprocessor to include header files in the program**
using namespace std;	using namespace statement just means that in the scope it is present, make all the things under the std namespace available without having to prefix std:: before each of them.
int main()	main() function is **the entry point of any C++ program**. When a C++ program is executed, the execution control goes directly to the main() function. Every C++ program have a main() function.
return 0;	A return 0 means that **the program will execute successfully and did what it was intended to do**.

Tokens in Program

- When the compiler is processing the source code of a C++ program, each group of characters separated by white space is called a **token**

- Tokens are the smallest individual units in a program.

- A C++ program is written using tokens. It has the following tokens:
  - Keywords
  - Identifiers
  - Constants
  - Strings
  - Operators

Keywords in C++

- Keywords(also known as reserved words) have special meanings to the C++ compiler and are always written or typed in short(lower) cases.

- There are 95 keywords in C++, of which around 30 are unavailable in the C language.

asm	auto	bool	break
case	catch	char	class
const	const_cast	continue	default
delete	do	double	dynamic_cast
else	enum	explicit	export
extern	false	float	for
friend	goto	if	inline
int	long	mutable	namespace
new	operator	private	protected
public	register	reinterpret_cast	return
short	signed	sizeof	static
static_cast	struct	switch	template
this	throw	true	try
typedef	typeid	typename	union
unsigned	using	virtual	void
volatile	wchar_t	while	

Some Commonly used keywords

VARIABLES

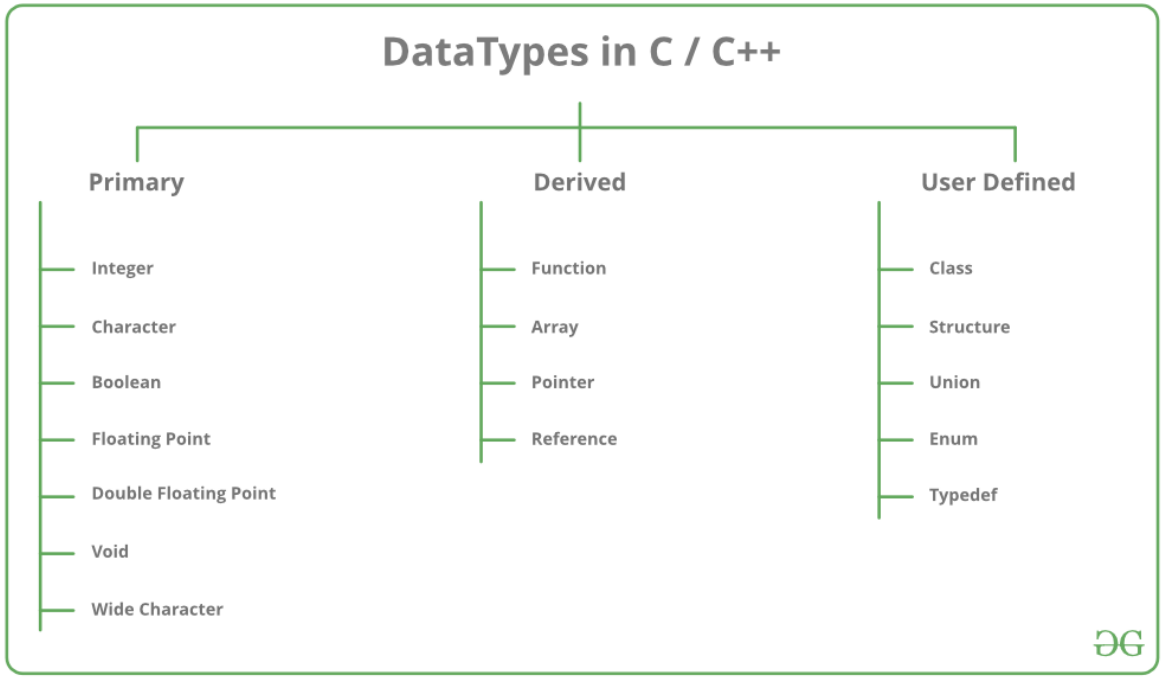
- Variables in C++ is a name given to a memory location. It is the basic unit of storage in a program.
- The value stored in a variable can be changed during program execution
- In C++, all the variables must be declared before use.

IDENTIFIERS

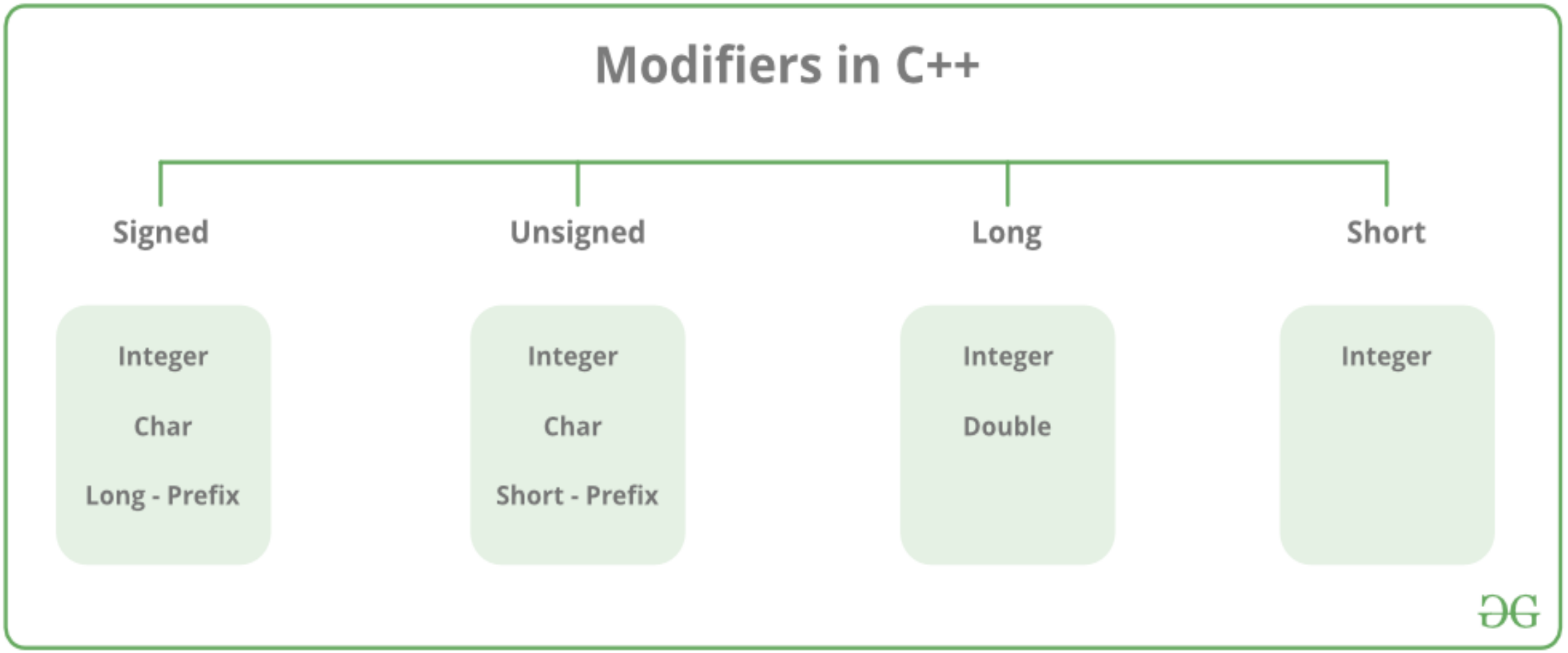
- Identifiers are used as the general terminology for the naming of variables, functions and arrays.
- Rules for naming Identifiers
  - They must begin with a letter or underscore(_).
  - They must consist of only letters, digits, or underscore. No other special character is allowed.
  - It should not be a keyword.
  - It must not contain white space.
  - It should be up to 31 characters long as only the first 31 characters are significant.
  - main: method name.
  - a: variable name.

Data Types in C++

- All variables use data type during declaration to restrict the type of data to be stored. Therefore, we can say that data types are used to tell the variables the type of data they can store.
- Whenever a variable is defined in C++, the compiler allocates some memory for that variable based on the data type with which it is declared.
- Every data type requires a different amount of memory.
- C++ supports the following data types:
  - Primary or Built-in or Fundamental data type
  - Derived data types
  - User-defined data types



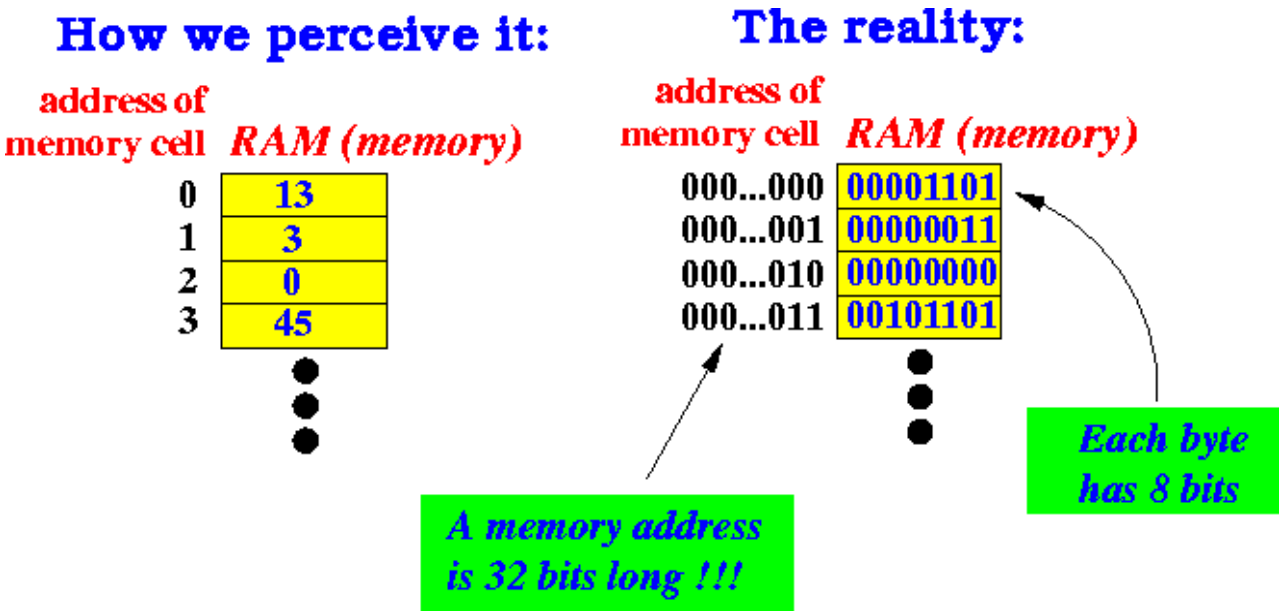
S.No	Type	Description
1	bool	Stores either value true or false.
2	char	Typically a single octet (one byte). This is an integer type.
3	int	The most natural size of an integer for the machine.
4	float	A single-precision floating point value.
5	double	A double-precision floating point value.
6	void	Represents the absence of type.


Modifiers in C++
Signed
Unsigned
Long
Short
Integer
Char
Long - Prefix
Integer
Char
Short - Prefix
Integer
Double
Integer

DataType	Size(in Bytes)	Range
int or signed int	4 Bytes	-2,147,483,648 to 2,147,483,647
unsigned int	4 Bytes	0 to 4,294,967,295
short int	2 Bytes	-32,768 to 32,767
long int	4 Bytes	-2,147,483,648 to 2,147,483,647
unsigned short int	2 Bytes	0 to 65,535
unsigned long int	8 Bytes	0 to 4,294,967,295
long long int	8 Bytes	-(2^63) to (2^63)-1
unsigned long long int	8 Bytes	0 to 18,446,744,073,709,551,615
signed char	1 Bytes	-128 to 127
unsigned char	1 Bytes	0 to 255
wchar_t	2 or 4 Bytes	1 wide character
float	4 Bytes	
double	8 Bytes	
long double	12 Bytes	

LANGUAGE OF COMPUTERS: BINARY

- Electronic devices can understand only 2 states – **ON** or **OFF** || **0** or **1**
- Data stored in the memory of computer is in the form of 1 or 0.
- Bit : The bit is a basic unit of information in computing. A bit stores just a 0 or 1.
- Byte: combination of 8 bits.



Input-Output in C++

- **Input**–data given by user to program in execution
  - Can be given in the form of file or from command line.
  - Set of built-in functionsto read input.
  - ***cin>>x;***
  - **x**is any variable that you might be using.

```
#include <iostream>
using namespace std;
int main()
{
    int x;
    cin>>x;
    return 0;
}
```

Input-Output in C++

- **Output**– It means to display some data on screen or file.
  - Can be given in the form of file or from command line.
  - Set of built-in functions to display output.
  - ***cout<<x;***
  - **x** is any variable that you want to print

```
#include <iostream>
using namespace std;
int main()
{
    int x=20;
    cout<<x;
    return 0;
}
```

OPERATORS

- Operator is a symbol to perform specific mathematical or logical functions.
- For eg '+' operator adds two operands.
- Types of operator :
  - Arithmetic Operators (+,-,/,*)
  - Relational Operators (<, >, <=, >=, ==)
  - Logical Operators (&&, ||)
  - Bitwise Operators (^, |, &, ~)
  - Assignment Operators (=)
  - Misc Operators
- Operators have a specific precedence and associativity which will be discussed later.

STATEMENTS

- In C++ program , instructions are written in the form of statements.
- Examples of statements :
  - int x = 5;
  - func(a,b);
  - x= y- z;
- Statements insides {} curly braces are known as compound statement.

COMMENTS

- In C++ program , comments are used to increase readability.
- Examples of comments:
- // this is a single line comment
- /* This is a
  multiline comment */

OPERATORS

Arithmetic	+, -, *, /, %
Assignment	=
Increment/Decrement	++, --
Relational	<, >, <=, >=, ==, !=
Logical	&&, ||, !
Conditional	? :
Comma	,
Bitwise	&, |, ~, <<, >>, ^

OPERATORS

Unary Operator: Single Operand

```cpp
#include <iostream>
using namespace std;
int main()
{
    int a;
    a = 10;
    cout<<a<<"\n";
    a = -a;
    cout<<a<<"\n";
    a = -a;
    cout<<a<<"\n";
}
```

Binary Operator: Two Operand

- Modulo only with integers

```cpp
#include <iostream>
using namespace std;
int main()
{
    int a, b, sum, diff, mul, quot, rem;
    cin>>a;
    cin>>b;
    sum  = a + b;
    diff = a - b;
    mul = a * b;
    quot = a / b;
    rem = a % b;
    cout<<sum<<"\n";
    cout<<diff<<"\n";
    cout<<mul<<"\n";
    cout<<quot<<"\n";
    cout<<rem<<"\n";
}
```

INCREMENT AND DECREMENT OPERATORS

Prefix: First operate then use

- y=++x;
  - x=x+1;
  - y=x;

- y=--x;
  - x=x-1;
  - y=x;

```
#include <iostream>
using namespace std;
int main()
{
    int x=8;
    cout<<x<<"\n";
    cout<<++x<<"\n";
    cout<<x<<"\n";
    cout<<--x<<"\n";
}
```

Postfix: First use then operate

- y=x++;
  - y=x;
  - x=x+1;

- y=x--;
  - y=x;
  - x=x-1;

```
#include <iostream>
using namespace std;
int main()
{
    int x=8;
    cout<<x<<"\n";
    cout<<x++<<"\n";
    cout<<x<<"\n";
    cout<<x--<<"\n";
}
```

RELATIONAL OPERATORS

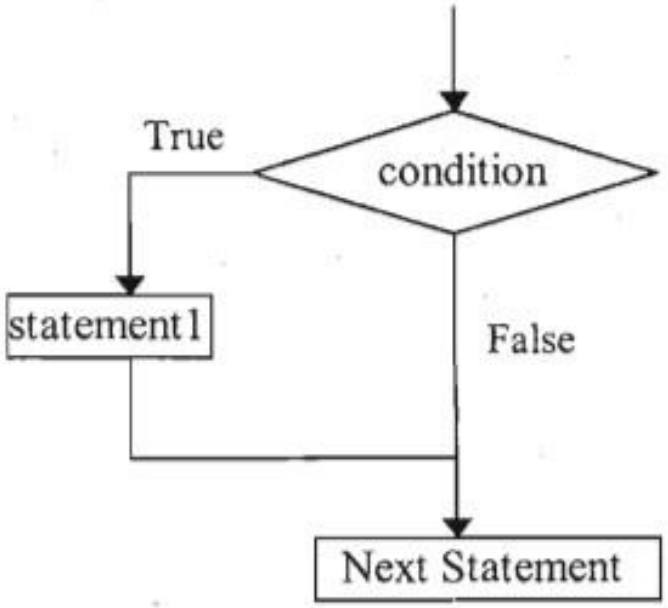
Operator	Meaning
<	less than
<=	less than or equal to
= =	equal to
!=	Not equal to
>	Greater than
>=	Greater than or equal to

Let's take a = 9 and b = 5

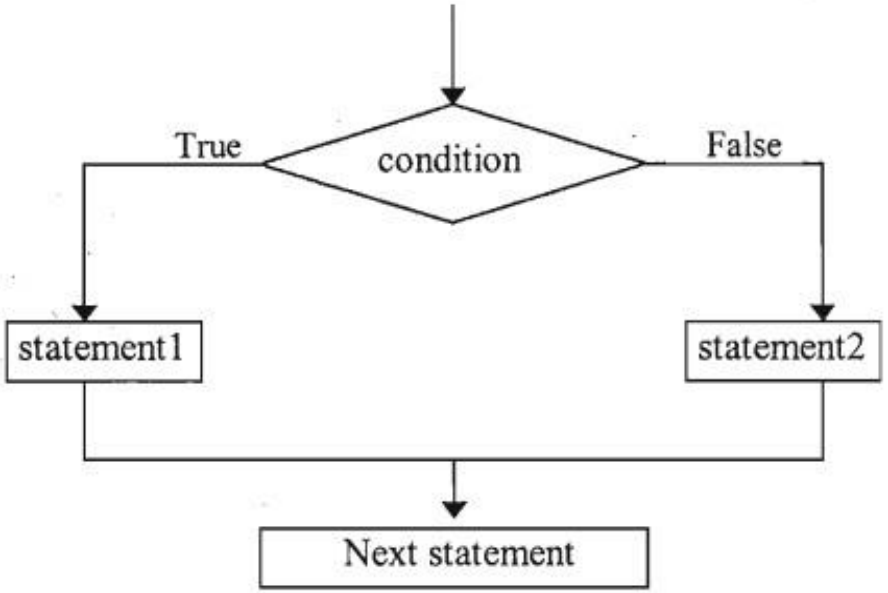
Expression	Relation	Value of Expression
a < b	False	0
a <= b	False	0
a= =b	False	0
a != b	True	1
a > b	True	1
a >= b	True	1
a = = 0	False	0
b!=0	True	1
a>8	True	1
2 > 4	False	0

IF ELSE CLAUSE

Why if else is needed?



Flow chart of if control statement



Flow chart of if…else control statement

```cpp
#include <iostream>
using namespace std;
int main()
{
    int num;
    cout<<"Enter a number : ";
    cin>>num;
    if(num < 0){
        cout<<"The number is negative. \n";
    }else{
        cout<<"The number is non-negative. \n";
    }
    return 0;
}
```

```cpp
#include <iostream>
using namespace std;
int main()
{
    int num;
    cout<<"Enter a number : ";
    cin>>num;
    if(num < 0){
        cout<<"The number is negative. \n";
    }
    return 0;
}
```

IF ELSE PROGRAMS

LOGICAL OPERATORS

- When we need to combine two or more conditions

Operator	Meaning
&&	AND
||	OR
!	NOT

A N D

Condition1	Condition2	Result
False	False	False
False	True	False
True	False	False
True	True	True

```cpp
#include <iostream>
using namespace std;
int main()
{
    int num;
    cout<<"Enter a number : ";
    cin>>num;
    if(num > 0 && num%2==0){
        cout<<"The number is a positive even. \n";
    }
    return 0;
}
```

O R

Condition1	Condition2	Result
False	False	False
False	True	True
True	False	True
True	True	True

```cpp
#include <iostream>
using namespace std;
int main()
{
    int num;
    cout<<"Enter a number : ";
    cin>>num;
    if(num > 0 || num == 0){
        cout<<"The number is either positive or zero. \n";
    }
    return 0;
}
```

NOT

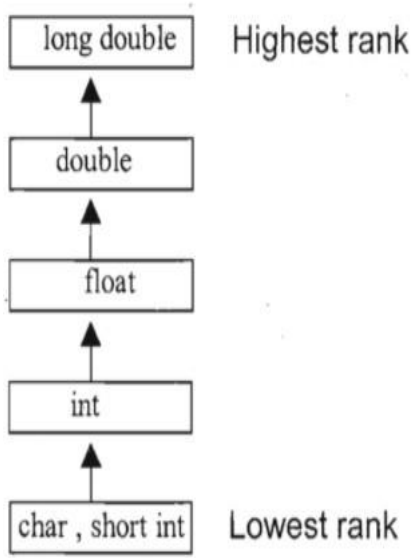
Condition	Result
False	True
True	False

```cpp
#include <iostream>
using namespace std;
int main()
{
    int num;
    cout<<"Enter a number : ";
    cin>>num;
    if(!(num==0)){
        cout<<"The number not a zero. \n";
    }
    return 0;
}
```

TYPE CONVERSION

Implicit: Done by compiler

- In case of operations done between different types of operators, lower rank is automatically converted into higher rank and result is also in higher rank
- In case of assignment LHS operator gets converted into data type of RHS




```
#include <iostream>
using namespace std;
int main()
{
    int i1 = 5, i2 = 3;
    float f1 = 2.5, f2 = -3.8;
    i1 = 80.56;
    cout<<i1<<"\n";
    f1 = i1 + f2;
    cout<<f1<<"\n";
    return 0;
}
```

Explicit: Done by Programmer

- Also known as type casting or coercion
- Cast operator: Unary Operator
- Syntax: (datatype)expression

```
#include <iostream>
using namespace std;
int main()
{
    int x=5, y=2;
    float p, q;
    p = x/y;
    cout<<p<<"\n";
    q = (float)x/y;
    cout<<q<<"\n";
    return 0;
}
```

(int)20.3	constant 20.3 converted to integer type and fractional part is lost(Result 20)
(float)20/3	constant 20 converted to float type, and then divided by 3 (Result 6.66)
(float)(20/3)	First 20 divided by 3 and then result of whole expression converted to float type(Result 6.00)
(double)(x +y -z)	Result of expression x+y-z is converted to double
(double)x+y-z	First x is converted to double and then used in expression

PRECEDENCE AND ASSOCIATIVITY

Precedence	Operator	Description	Associativity
1	`::`	Scope resolution	Left-to-right

2	`++ --`
	`()`
	`[]`
	`.`
	`->`
		Suffix/postfix increment and decrement
		Function call
		Array subscripting
		Element selection by reference
		Element selection through pointer	

3	`++ --`
	`+ -`
	`! ~`
	`(`*type*`)`
	`*`
	`&`
	`sizeof`
	`new, new[]`
	`delete, delete[]`
		Prefix increment and decrement
		Unary plus and minus
		Logical NOT and bitwise NOT
		Type cast
		Indirection (dereference)
		Address-of
		Size-of
		Dynamic memory allocation
		Dynamic memory deallocation
			Right-to-left

4	`.* ->*`	Pointer to member	Left-to-right
5	`* / %`	Multiplication, division, and remainder	
6	`+ -`	Addition and subtraction	
7	`<< >>`	Bitwise left shift and right shift	
8	`< <=` `> >=`	For relational operators < and ≤ respectively For relational operators > and ≥ respectively	
9	`== !=`	For relational = and ≠ respectively	
10	`&`	Bitwise AND	
11	`^`	Bitwise XOR (exclusive or)	
12	`|`	Bitwise OR (inclusive or)	
13	`&&`	Logical AND	
14	`||`	Logical OR	

15	`?:`
	`=`
	`+= -=`
	`*= /= %=`
	`<<= >>=`
	`&= ^= |=`
		Ternary conditional
		Direct assignment (provided by default for C++ classes)
		Assignment by sum and difference
		Assignment by product, quotient, and remainder
		Assignment by bitwise left shift and right shift
		Assignment by bitwise AND, XOR, and OR
			Right-to-left

16	`throw`	Throw operator (for exceptions)	
17	`,`	Comma	Left-to-right