1.2 write your first prog + operators

Made By :-
Aditya Jain

Why do we need programming lang?

A prominent purpose of programming lang is to provide instructions to a computer.

Is C++ is a high-level lang?

C++ is regarded as a middle-level lang, as it comprises a combination of both high-level & low-level lang feature. It is a superset of C & that virtually any legal C prog is a legal C++ program.

* The code is portable & the syntax is human-readable. For these reason, C & C++ are HLL.

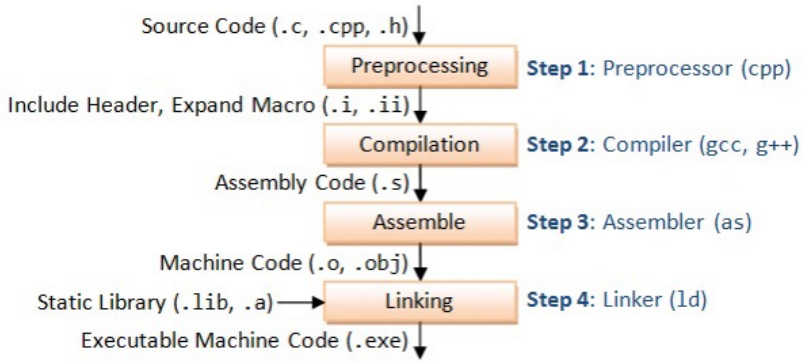
Why use C++?

→ C++ is one of the world's most popular prog. lang. that can be found in today's OS, GUI & embedded systems.

→ It is object-oriented prog. lang. which give us a clear structure to program & allows code to be re-used, lowering development cost.

* The main difference b/w C & C++ is that C++ supports classes & objects, while C does not.

How Compiler compile the code :-



Where to start the code :-

Code starts from :-

```
int main ()
```

→ function

only run when it is called

a block of code that together perform a task

return-type : integer

function name

```
int main ()
{

}
```

syntax

block of code

↳ means all the code inside that belongs to the main function

first C++ program

header file library :- tell us work with input & output objects, such as cout.
+ header file add functionality to c++ prog.

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

means that we can use names for objects & variables from the standard library.

Pronouce as See-out : is an object with the insertion operator << to output / print text

cannot use without header file

String ↓ Sequence of character

end of line

return 0 in main function means that the program executed successfully

Use-case	return 0	return 1
In the main function	return 0 in the main function means that the program executed successfully.	return 1 in the main function means that the program does not execute successfully and there is some error.
In user-defined function	return 0 means that the user-defined function is returning false.	return 1 means that the user-defined function is returning true.

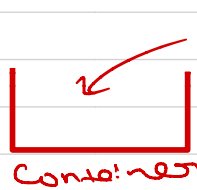
Without using namespace std; cout work?

Yes, cout work without using standard namespace. But we have to write std:: before cout.

```
std :: cout << " Aditya Jain " ;
```

Variables :- a container that is used to store the data values.

→ value stored in a variable can be changed during program execution



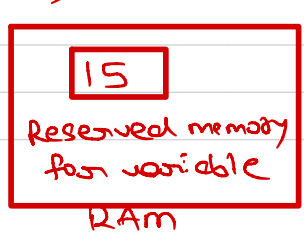
→ A variable is only a name given to a memory location, all the operations done on the variable effects that memory location

How to declare single variable

Syntax :- type variable_name ;

for ex int age = 15 ;

- int → data type → Explain in data types
- age → variable name
- = → assignment operator
- 15 → value

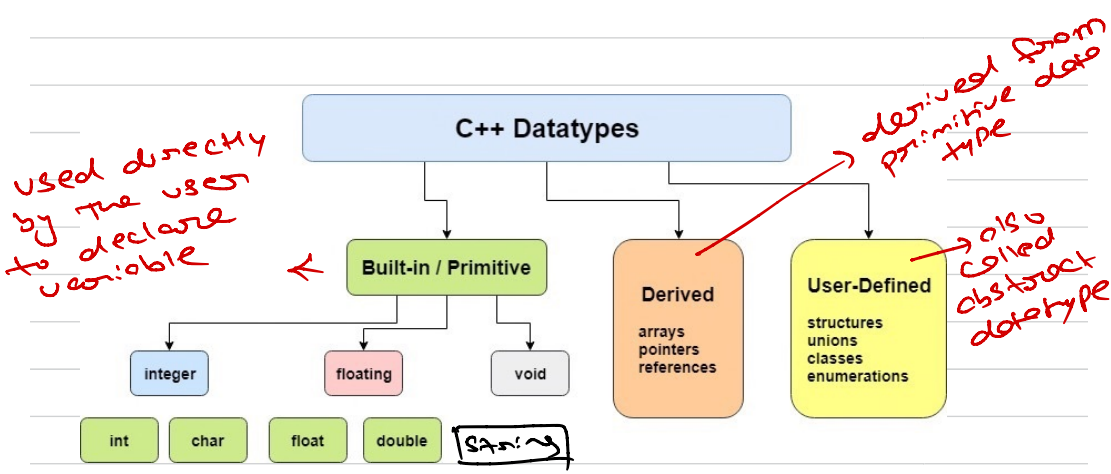


How to declare multiple variable

Syntax type variable-name1 , variable-name2;

* A variable name can consist of alphabets (both upper & lower case), number, & the underscore '_' character.
However, the name must not start with a number. Or special characters.

Data types → Specifies the size & type of information the variable will store




```
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";     // String
```

Basic Data types

Data Type	Size	Description
`boolean`	1 byte	Stores true or false values
`char`	1 byte	Stores a single character/letter/number, or ASCII values
`int`	2 or 4 bytes	Stores whole numbers, without decimals
`float`	4 bytes	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 15 decimal digits

Datatype with their size & range

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

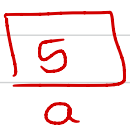
How data is stored ?

Signed (-ve → +ve)
Unsigned (+ve)

Data are stored in the memory in the form of binary.

UnSigned Data values → It can hold positive number & zero. 0 → +ve

for Ex int a = 5 ;



← In actual, values are not stored like this in memory.

In memory, it is stored the binary of 5.

Binary :- numbers & values are represented by only two digits → 0 and 1.
Binary is base-2, meaning that is only uses two digits or bits.

In the above Ex int a = 5 ; where a is a integer data type which contain the 4 bytes. i.e. 32 bits.

Another Ex :-

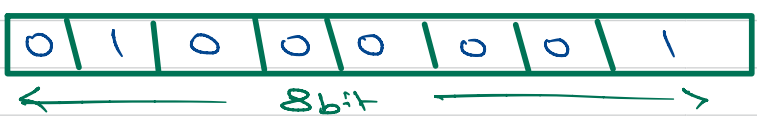
```
char letter = 'A';
```

Each character is mapped with ascii value.

for Ex A $\xrightarrow[\text{with}]{\text{mapped}}$ 65

Binary of 65 $\Rightarrow$ 1000001

char $\rightarrow$ 1 byte $\rightarrow$ 8bit $\rightarrow$ $\underbrace{0-255}_{\text{range}}$



```
bool flag = true;
```

1 byte $\rightarrow$ 8 bits

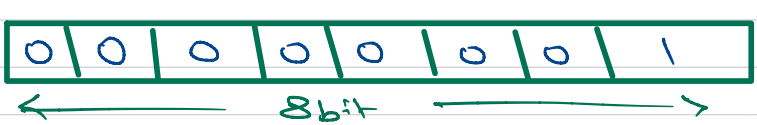
bool

true $\rightarrow$ 1 | false $\rightarrow$ 0

It needs only 1 bit

why bool datatype contain 8 bits in memory ?

Smallest space addressable in memory is 1byte. that's why bool contain 1 byte memory space



Signed Data values → It can hold negative, positive & zero -ve ↔ 0 ↔ +ve

-ve values are stored same as +ve values ? NO

There are three steps to stored the negative values :-

1. Ignore the -ve sign
2. find binary equivalent
3. takes 2's compliment

It is a mathematical operation to reversibly convert a +ve binary number into a -ve binary number with equivalent -ve value.

How to make 2's compliment :-

1's compliment
adding 1

inverting or flipping all bits — changing every 0 → 1 & every 1 → 0.

```
int a = -5;
```

(1) Ignore the -ve sign

(2) binary of a in 32 bit

00000000 00000000 00000000 00000101

(3) Convert 2's compliment

(3.1) 1's compliment

11111111 11111111 11111111 11111010

(3.2) Adding 1

```
  11111111 11111111 11111111 11111010
                                   + 1
  ------------------------------------
  11111111 11111111 11111111 11111011
```

↓

binary of -5 in memory

* binary converted into -ve decimal number with the process
  - 2's compliment
  - binary to decimal conversion
  - adding negative sign

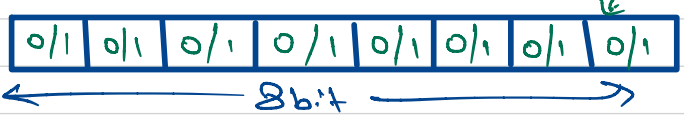
formula → $0 \rightarrow 2^n - 1$ where n = bits

Ranges of Data types → unsigned data

1) unsigned char → 1 byte → 8 bits

Total combination = $2^8 \Rightarrow 256$

why use 2^8 ?



Range ⇒ 0 — 255

OR

0 — $2^8 - 1$

why -1 ?

because the range calculated from 0

2) unsigned int → 4 byte → 32 bits

total combination = 2^{32}

range ⇒ 0 — 4294967295

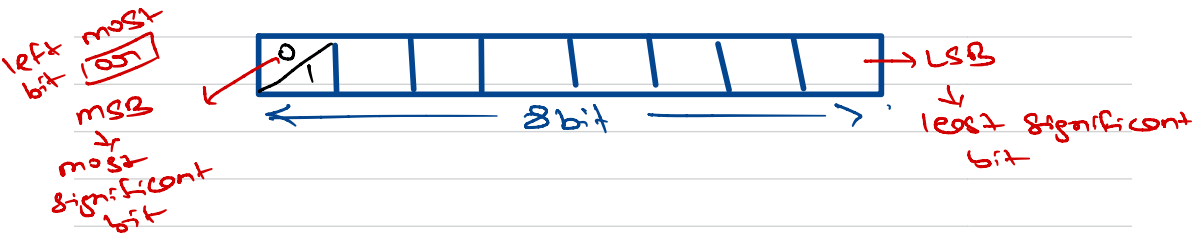
OR

0 — $2^{32} - 1$

-ve — 0 — +ve

Ranges of Data types → Signed data

formula :- $-2^{n-1} \rightarrow 2^{n-1} - 1$



MSB works like the **sign bit** → 0 → +ve, 1 → -ve

MSB bit tell us whether the number is -ve or +ve.

In Signed data we does not consider the MSB.

therefore, we have only 7 bits.

1) char → 1byte → 8 bits

+ve =  $\Rightarrow 2^7$

-ve =  $\Rightarrow 2^7$

range :- $-2^7 \rightarrow 2^7 - 1 \Rightarrow -128 \rightarrow 127$

$-128 \leftarrow -1 \quad 0 \rightarrow 127$

2) Int → 4 byte → 32 bit

+ve range ⇒ $0 \rightarrow 2^{n-1} - 1$

$0 \rightarrow 2^{32-1} - 1$

$0 \rightarrow 2^{31} - 1$

-ve range ⇒ $-2^{n-1} \rightarrow -1$

$-2^{32-1} \rightarrow -1$

$-2^{31} \rightarrow -1$

Combined range ⇒ $-2^{31} \rightarrow 2^{31} - 1$

* How to check the datatype size in C++ program?

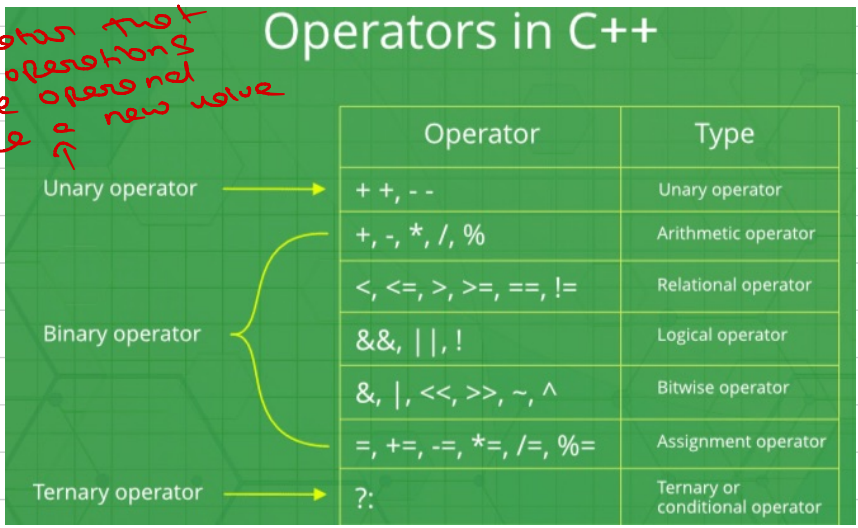
using sizeof() operator we can check the size of data.

Syntax: sizeof (dataType);

Operators

Operator is a symbol that operates on a value to perform specific mathematical & logical computations

the operator that perform operations on single operand to produce a new value



Arithmetic Operator :- used to perform 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 by 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

2) Assignment operators :- used to assign the values to the variable

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

3) Comparison operators :- used to compare the values or variables

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

4) **Logical operators :-** is used to determine the logic between values or variables

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)

5) **Bitwise operators :-** operate at bitwise level

If both bits are 1 then 1 otherwise 0 — If any bit are 1 then 1 otherwise 0 — If both bit are same then 0 otherwise 1

X	Y	X & Y AND	X | Y OR	X ^ Y XOR
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0

to cancel duplicate numbers we use XOR

XOR → 2 ^ 3 ^ 4 ^ 3 ^ 2 ⇒ 4 o/p

1) Bitwise AND → 12 & 25 ⇒ ?

12 → 00001100
25 → 00011001
} Binary

```
  00001100
& 00011001
----------
  00001000   ⇒ 8 (In decimal)
```

2) Bitwise OR → 12 | 25 ⇒ ?

```
  00001100
| 00011001
----------
  00011101   ⇒ 29 (In decimal)
```

3) Bitwise XOR → 12 ^ 25 ⇒ ?

```
  00001100
^ 00011001
----------
  00010101   ⇒ 21 (In decimal)
```

cout << 5 ^ -5 ; ⇒ o/p ⇒ -2

5 ⇒ 00000101
-5 ⇒ ^ 11111011 (2's complement)

11111110 ⇒ 0000 0010 ⇒ -2

msB ⇒ -ve (2's complement)

also known as bitwise complement

~a , ~(a) , (~a) → some result

4) **Bitwise NOT** → works only one operand & changes binary digits 1 → 0 and 0 → 1

* Bitwise NOT of any integer N is equal to −(N+1) in signed value

for ex: Bitwise NOT of 35 → −(35+1) ⇒ −36

35 = 00100011 (In binary)

~ 00100011

11011100

MSB = 1
It convert into neg number but binary → decimal we take 2's compliment

11011100
↓ 1's compliment
00100011
↓ 2's compliment
00100100 ⇒ −36

```
bool num = 1;
cout << (~num);
o/p = -2
```

When we cout it take as an integer value

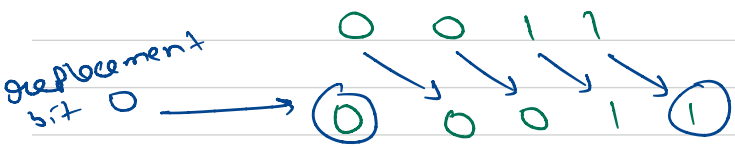
```
bool num = 1;
bool ans = (~num);
cout << ans;
o/p = 1
```

we store the o/p in bool value & except 0 all values are 1 in bool

>>

→ divisible by 2^n when n = input right shift bit

5) **Right shift operator** :- shift all bits towards the right by a certain number of specified bits.

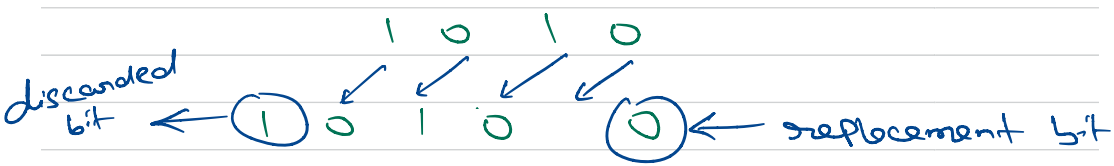


2 >> 1 => 0010 => 0001 => 1

-1 >> 1 => -1 ?

→ multiply by 2^n

6) **Left shift operator** :- shift all bits towards the left by a certain number of specified bit.



10 << 1 => 5

-2 << 1 = -4 + warning

---

* when we the value by -ve number it throws the garbage value & shift more than the size of the integer also throw garbage

```
int x = -5;
x = x >> 1;
cout << x << endl;
```

} o/p = -3

$-5 \Rightarrow 11111101 \quad >> 1$

when we right shift the neg number the MSB bit is preserve & mainted

→ used on int, char, pointer, float, etc

6) Pre & post increament/Decrement

↓
unary operator → incr/decr values by 1

Prefix (++m) (--m)
first incr/decr the value then use the variable

postfix (m++) (m--)
first use the variable then incr/decr the value

* The Post-incr have higher precedence that pre-incr as it is postfix operator while pre-incr comes in unary operator

```
int a = 5;
int b = 5;

cout << (++a);   => 6
cout << (b++);   => 5
```

Operator Precedence & Associativity

Precedence	Operator	Description	Associativity
1	`::`	Scope resolution	Left-to-right
2	`a++` `a--`	Suffix/postfix increment and decrement	
	type`()` *type*`{}`	Functional cast	
	`a()`	Function call	
	`a[]`	Subscript	
	`.` `->`	Member access	
3	`++a` `--a`	Prefix increment and decrement	Right-to-left
	`+a` `-a`	Unary plus and minus	
	`!` `~`	Logical NOT and bitwise NOT	
	(*type*)	C-style cast	
	`*a`	Indirection (dereference)	
	`&a`	Address-of	
	`sizeof`	Size-of[note 1]	
	`new` `new[]`	Dynamic memory allocation	
	`delete` `delete[]`	Dynamic memory deallocation	
4	`.*` `->*`	Pointer-to-member	Left-to-right
5	`a*b` `a/b` `a%b`	Multiplication, division, and remainder	
6	`a+b` `a-b`	Addition and subtraction	
7	`<<` `>>`	Bitwise left shift and right shift	
8	`<=>`	Three-way comparison operator (since C++20)	
9	`<` `<=`	For relational operators < and ≤ respectively	
	`>` `>=`	For relational operators > and ≥ respectively	
10	`==` `!=`	For relational operators = and ≠ respectively	
11	`&`	Bitwise AND	
12	`^`	Bitwise XOR (exclusive or)	
13	`|`	Bitwise OR (inclusive or)	
14	`&&`	Logical AND	
15	`||`	Logical OR	
16	`a?b:c`	Ternary conditional[note 2]	Right-to-left
	`throw`	throw operator	
	`=`	Direct assignment (provided by default for C++ classes)	
	`+=` `-=`	Compound assignment by sum and difference	
	`*=` `/=` `%=`	Compound assignment by product, quotient, and remainder	
	`<<=` `>>=`	Compound assignment by bitwise left shift and right shift	
	`&=` `^=` `|=`	Compound assignment by bitwise AND, XOR, and OR	
17	`,`	Comma	Left-to-right