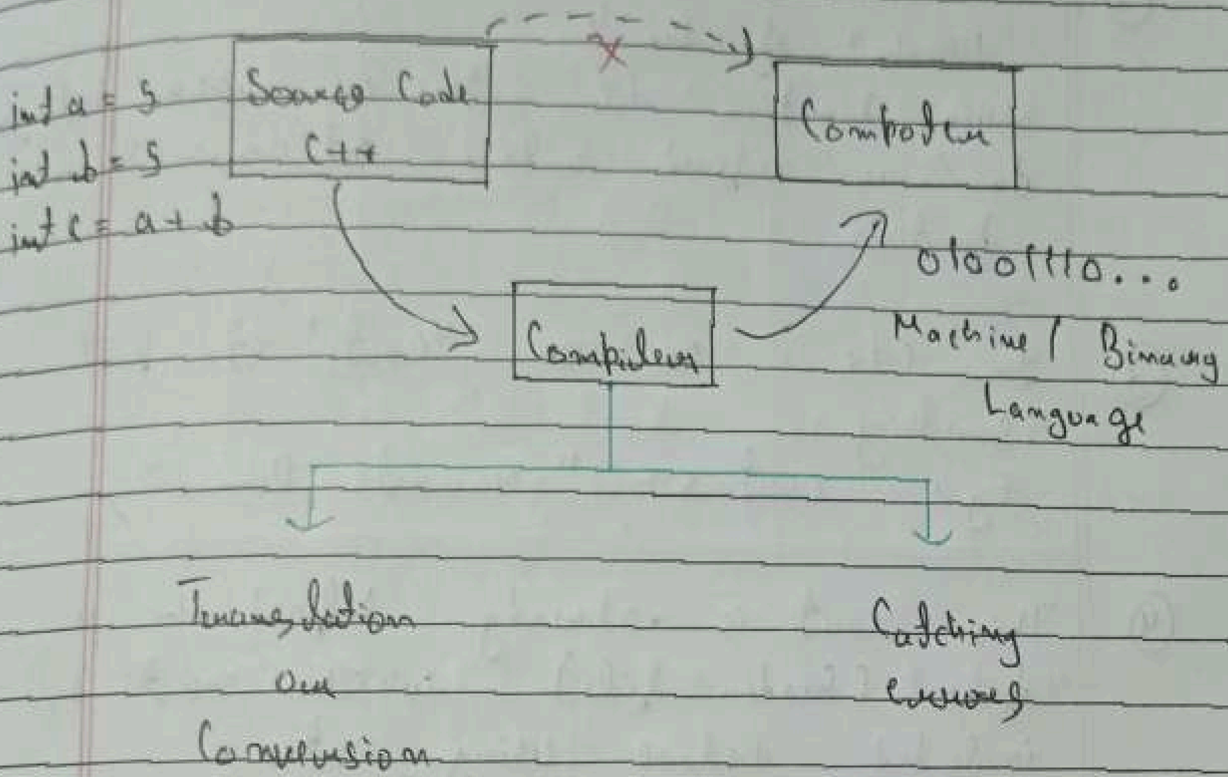
First Program in C++



IDEs (Integrated Development Environment):

An environment that helps you write, run and even debug (in some cases) code in a programming language.
Eg. VS Code, Code Studio, Eclipse, NetBeans etc

First Program in C++ : (Namaste Duniya)

① Our Program will find int main() { and start executing }

② :- int main () { These brackets show the 'scope' of the }
main function i.e. the code which belongs to / is defined within int main() function.

③ In C++, we use 'cout' to print something.
Eg :- cout << "Namaste Duniya";

④ This cout is already defined in a file (header file) which must be included before using cout.

⇒ # include <file name> is a Preprocessor directive which runs before the program is compiled and includes the file to be used later in the source code. A file called iostream has cout defined in it so:
# include <iostream>
Hint: i/o means input / output

⑤ Name spaces : Stack overflow Question
using namespace std;

⑥ Using cout :
we use '<<' after 'cout' to display something to standard output (your screen) within std namespace.

⑦ endl : Used to enter new line. Just like [Enter], endl is like "\n" which is a new line escape sequence character used in various languages including C++.
cout << "Namaste Duniya" < endl;

⑧ endl ; → ; is used to terminate statements.

Data Types : Different types of data to be stored memory, Eg : integer, float, character, etc.

Eg : int : stores integer like -5, 0, 8 etc
char : stores character like 'a', '+', '$' ; '7', etc
float : Floating point values like -2.014, 1.000, 6.7 etc

Different data types use different amount of memory used also depends on the architecture of

Data type	Meaning	Size (in Bytes)
int	integer	2 or 4
float	Floating Point	4
double	Double floating Point	8
char	Character	1
wchar_t	wide Character	2
bool	boolean	1
void	Empty	0

Character : A 1-byte (= 8 bits) data type that takes character.

```
Char ch = 'a';
```

Boolean : True / False. Take 1 bit and
1 : True
0 : False

```
bool isGood = 1;
bool isBad = false;
```

Float / Double : Float takes 4/8 bytes.
Double takes 8 bytes

```
float num1 = 1.2;
double num2 = 2.4;
```

Variable naming / Nomenclature :

1 :- Can contain alphabets, numbers and underscores.

2 :- Cannot start with a number

3 :- Cannot be keywords like int, cout, double, bool,

4 :- Case Sensitive

5 :- Cannot contain special symbols like %, $, !, #.

Warning :- Don't use same variable type multiple times.

Check the size of different data types for your system :

sizeof (variable name);

How is Data stored in Memory?

Eg :- int a = 8 ; // int takes 4 bytes = 32 bits

In binary, 8 = 1000 (4 bits needed)

∴ int a →

00000000 00000000 00000000 00001000 ← 32 bits

Eg : int b = 5;

$\overset{b}{\boxed{5}}$ 4 bytes

address = 100 (assume

$\underbrace{100, 101, 102, 103}$

4 bytes are consumed

Eg : char c = 'a'

characters are mapped to the standard ASCII values

'a' → 97	'A' → 65
'b' → 98	'B' → 66
'c' → 99	'C' → 67
⋮ ⋮	⋮ ⋮
'z' → 122	'Z' → 90

ASCII Table

Homework

char c = 'a';

$\xrightarrow[\text{ASCII}]{}$ 97 $\xrightarrow{\text{Binary}}$ 1100001 → store

$\underbrace{01100001}$

Type Casting and Storing Variables

Conversion of one data type to another (if it is valid) is called type casting.

Ex : int a = 'a';
variable a will store the ASCII value of 'a' = 97

Ex : char ch = 98;
98 will automatically get typecasted to its corresponding character.

This automatic typecasting is called
Implicit typecasting.

What if we type cast from int to char but the value is too large to be

Soln : A warning is thrown and the last byte from the original value is given to the character.

How are negative number stored?

Soln : The first bit tells us whether the number is positive or negative

First Bit → 0 means positive

Steps to store -5 in binary format

① Ignore the -ve sign (5)
② Write the binary representation of 5
③ Take its 2's complement and store it.

Example :- a = -5

① -5 → 5 (ignore the -ve sign)
② 5 → 0101 → $\underbrace{00\ldots\ldots0101}_{\text{29 zeros}}$ (Binary)

③ 2's Complement = 1's Complement +1
5 → 00....0101
1's compl → 11....1010 (flip the bits)

2's Compl → $\underbrace{11\ldots\ldots1011}_{\text{29 Once}}$

<u>Displaying Negative Numbers</u> :-

① Take 2's Complement of the stored number.

Stored :- 111....1011

This shows -ve

11.... 1011

1's comp 00.....0100

2's comp 00.....0101 = ⑤ → -5 Print ho gaya!

Why 2's complement ?

If we store number as it is without using 2's complement, then

1	0	0	-----	0	0

and

0	0	0	---	0	0

will be saved & thus wast space.

Store only Positive Integer

The default signed representation allows us to store both positive & negitive values.

To store only positive integers, we use unsigned.

Eg : unsigned int a = 10;

What if we store a negative value in a unsigned number?

Ex: unsigned int a = -112;
    cout << a << endl;

Output:

4294967184 ??

Explanation:

We tried to store -112

-112 = 2's complement of 112

$112 = \underbrace{00......0}_{25 \text{ zeros}}1110000.$

1's compl. = 11......10001111
+ 1

2's compl. = 11.....10010000

Unsigned int uses all 32 bits to store the value and the MSB (=1) will make the value.

An unsigned int does not use the 2's complement again to display the number.

Thus, $\underbrace{11.....}_{25 \text{ ones}}10010000$. gets printed as it is in decimal.

There fore, output:
4294967184

Operators

Basic Arithmetic Operators :

+ , - , * , / , %

caution ⚠

① int / int = int (Floor value of answer)

Ex: 5/2 = 2
3/5 = 0
9/2 = 4

② int / float } float
float / int

double / int } double
int / double

cout << 5.0/2 << endl
Output : 2.5

Relational Operators :

== , >= , <= , > , < , !=

① is a = b ?

$a == b \xrightarrow{Yes} 1$
$\xrightarrow[No]{} 0$

(3) is a greater than or equal to b ?

$$\begin{tikzcd} a >= b \arrow[r, "yes"] \arrow[dr, "No"'] & 1 \\ & 0 \end{tikzcd}$$

Logical Operators : && , || , !
(AND) (OR) (NOT)

(1) Logical AND

All conditions must be true for the output to be true.

Ex : int a = 5, b = 10, c = 15 ;

```
cout << ((a>0) && (b!=0) && (c<=15));
```

Output = 1

(2) Logical OR

At least one condition must be true for the output to be true.

Ex : int a = 5, b = 10, c = 15;

```
cout << ((a>5) || (b<10) || (c>=15));
```

Output = 1

(3) Logical NOT

Inverts the logic. True ⇌ False

Non-zero ⇌ Zero

Ex :
```
int a = 10 , b = 0 ;
cout << (!a) << endl ;
cout << (!b) << endl ;
```

Output :
```
0
1
```