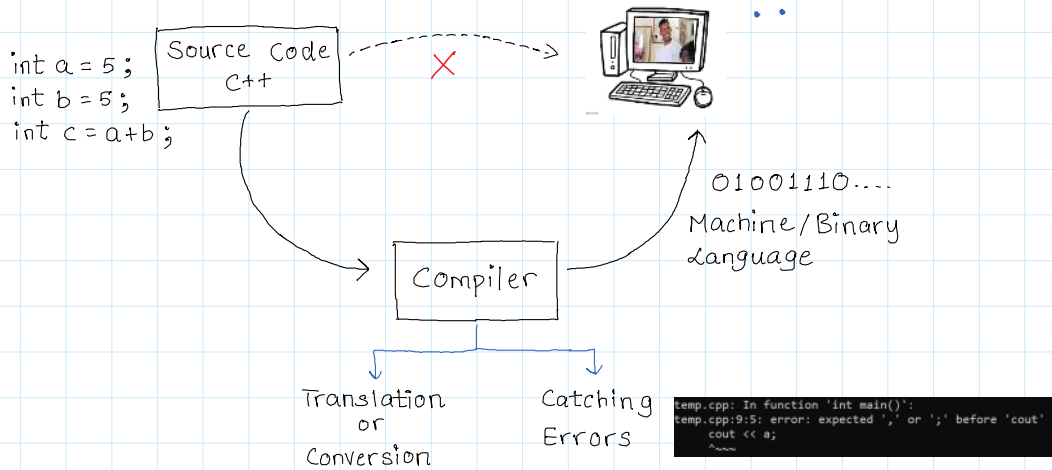


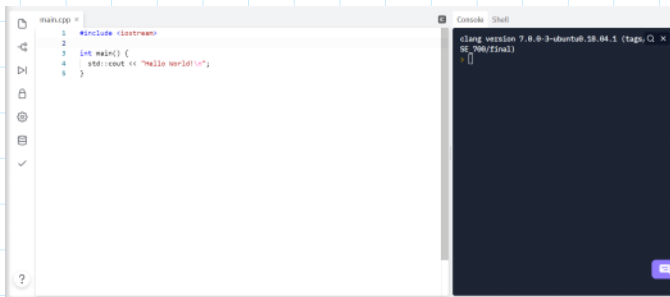


As discussed earlier :



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

[More about scope of a 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>`

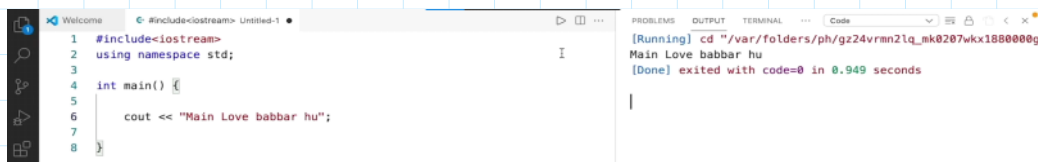
Hint: `i/o` means input/output.

⑤ Namespaces : [Stack Overflow Question](#)

`using namespace std;`

⑥ Using `cout` :

We use `<<` after `'cout'` to display something to Standard Output (your screen) within `std` namespace.

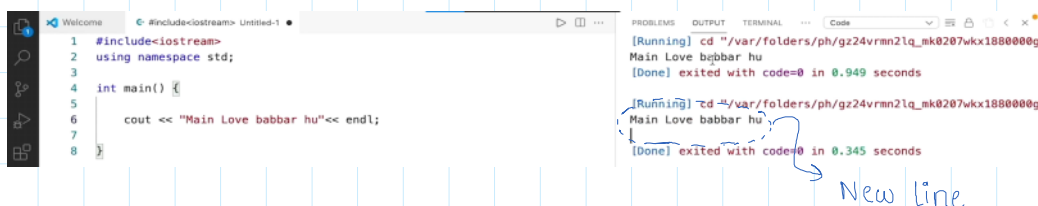


```
1 #include<iostream>
2 using namespace std;
3
4 int main() {
5
6     cout << "Main Love babbar hu";
7
8 }
```

PROBLEMS OUTPUT TERMINAL Code

[Running] cd "/var/folders/ph/gz24vrnn2lq\_mk0207wx1880000g" Main Love babbar hu [Done] exited with code=0 in 0.949 seconds

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;`



```
1 #include<iostream>
2 using namespace std;
3
4 int main() {
5
6     cout << "Main Love babbar hu" << endl;
7
8 }
```

PROBLEMS OUTPUT TERMINAL Code

[Running] cd "/var/folders/ph/gz24vrnn2lq\_mk0207wx1880000g" Main Love babbar hu [Done] exited with code=0 in 0.949 seconds

[Running] cd "/var/folders/ph/gz24vrnn2lq\_mk0207wx1880000g" Main Love babbar hu [Done] exited with code=0 in 0.345 seconds

New line

⑧ `endl;` ; is used to terminate statements.

**DATA TYPES :** Different types of data to be stored in memory. Eg- integer, float, character, double, etc.

Eg- `int` : Stores integers like -5, 0, 8, etc.

`char` : Single character like 'a', '+', '\$', '7', etc.

`float` : Floating point values like -2.014, 1.0000, 6.7800

Different data types use different amounts of memory. Amount of memory used also depends on the architecture of your CPU.

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

[Source: Programiz](#)

Character : A 1-byte (= 8 bits) data type that takes 1 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;
```

[Know more about C++ Data Types](#)

## 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, etc.
- ④ Case sensitive
- ⑤ Cannot contain special symbols like %, \$, !, #, etc.

WARNING: Don't use same variable name multiple times.

```
6 int a = 123;
7
8 cout << a << endl;
9
10 char a = 'v';
11
12 cout << a << endl;
```

```
11g] cd "/var/folders/ph/gz24vrnn2lq_mk0207wx1880000gn/T/"
CodeRunnerFile.cpp:10:10: error: redefinition of 'a' with a
char a = 'v';
^
CodeRunnerFile.cpp:6:9: note: previous definition is here
int a = 123;
^
```

Using different data types in code :



ASCII → 97 → Binary → 1100001 → Store

0 1 1 0 0 0 1  
1 byte



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

Example:

```
int a = 'a';
```

variable a will store the ASCII value of 'a' = 97.



Example:

```
char ch = 98;
```

98 will automatically get typecasted to its corresponding character.

This automatic typecasting is called **implicit typecasting**.

```
int a = 'a';
cout << a << endl;

char ch = 98;
cout << ch << endl;
```



```
[Running] cd "/var/folders/ph/gz24vrn2lq_mk0207wkx1880000g
97
b
```

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

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

```
char ch1 = 123456;
cout << ch1 << endl;
```



```
[Running] cd "/var/folders/ph/gz24vrn2lq_mk0207wkx1880000g
tempCodeRunnerFile.cpp:33:16: warning: implicit conversion
char ch1 = 123456;
~~~~~
1 warning generated.
```

How are negative numbers stored?

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

First Bit → 0 means Positive  
              → 1 means Negative

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 → 00.....0101 (Binary)  
                                29 Zeros

③ 2's complement = 1's complement + 1.

5 → 00.....0101

1's compl. → 11.....1010 (Flip the bits)

+                                  1

2's Compl. → 11.....1011

$$\begin{array}{r}
 + \quad \quad \quad 1 \\
 \hline
 2's \text{ Compl.} \rightarrow \underbrace{11 \dots 1011}_{29 \text{ Ones}}
 \end{array}$$

### Displaying Negative Number :

① Take 2's complement of the stored number

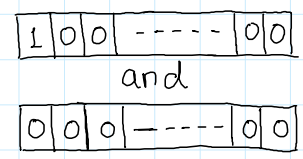
Stored : ~~11~~ 11 ..... 1011  
 This shows -ve

$$\begin{array}{r}
 11 \dots 1011 \\
 1's \text{ comp. } 00 \dots 0100 \\
 2's \text{ comp. } 00 \dots 0101 = 5
 \end{array}$$

-5 print ho gaya!

### Why 2's Complement ?

If we stored numbers as it is without using 2's complement, then



will be equal & thus waste space.



### Store only Positive Integers

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

To store only positive integers, we use unsigned.

Eg: unsigned int a = 10;

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

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

Output:  
 4294967184 ??

### Explanation :

We tried to store -112.

-112 = 2's Complement of 112.

$$112 = \underbrace{00 \dots 01110000}_{25 \text{ Zeros}}$$

$$\begin{array}{r}
 1's \text{ Compl.} = 11 \dots 10001111 \\
 + \quad \quad \quad 1 \\
 \hline
 \end{array}$$

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, 11...10010000. gets printed as it is in decimal.  
25 ones

[illegible]

Therefore,

```
unsigned int a = -112;

cout<< a << endl;
```

```
[Running] cd "/var/folders/ph/gz24vrnm2lq_mk0207wx1880000g
4294967184
```





## Basic Arithmetic Operators:

$+$ ,  $-$ ,  $*$ ,  $/$ ,  $\%$

### Caution

①  $\text{int} / \text{int} = \text{int}$  (Floor value of answer)

Examples:  $5/2 = 1$ .

$3/5 = 0$

$9/2 = 4$

②  $\text{int} / \text{float} \}$  float  
 $\text{float} / \text{int}$   
 $\text{double} / \text{int} \}$  double  
 $\text{int} / \text{double}$

`cout << 5.0/2 << endl;`

Output:

2.5

```
42 float a = 2.0/5;
43
44 cout << a << endl;
45
46 cout << 2.0/5 << endl;
47
```

```
[Running] cd "/var/folders/ph/gz24vrnm2lq_mk0207wx1880000g
0.4
0.4
[Done] exited with code=0 in 0.591 seconds
```

## Relational Operators :

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

① Is  $a = b$  ?

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

② Is  $a$  greater than or equal to  $b$  ?

$a >= b$   $\xrightarrow{\text{Yes}}$  1  
 $\xrightarrow{\text{No}}$  0

```
49
50 int a = 2;
51 int b = 3;
52 bool first = (a==b);
53 cout << first << endl;
54
55 bool second = (a>b);
56 cout << second << endl;
57
58 bool third = (a<b);
59 cout << third << endl;
60
61 bool fourth = (a>=b);
62 cout << fourth << endl;
63
64 bool fifth = (a<=b);
65 cout << fifth << endl;
66
67 bool sixth = (a!=b);
68 cout << sixth << endl;
69
```

```
[Running] cd "/var/folders/ph/gz24vrnm2lq_mk0207wx1880000g
0
0
1
0
1
1
[Done] exited with code=0 in 1.124 seconds
```

## Logical Operators :

$\&\&$ ,  $\|\|$ ,  $!$   
 (AND) (OR) (NOT)

① Logical AND :

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

Example : `int a = 5, b = 10, c = 15;`

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

Output:

1

## ② Logical OR :

At least 1 condition must be true for the output to be true.

Example: `int a = 5, b = 10, c = 15;`

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

Output:

1

## ③ Logical NOT :

Inverts the logic.  $\text{True} \Rightarrow \text{False}$

$\text{Non-zero} \Rightarrow \text{Zero}$

Example: `int a = 10, b = 0;`

```
cout << (!a) << endl;
```

```
cout << (!b) << endl;
```

Output:

0

1

Point out what was not covered in Typecasting and try to code yourself.