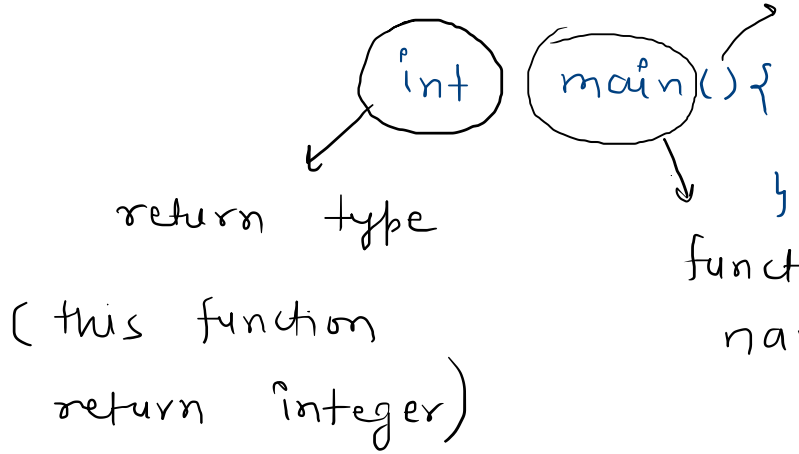
First c++ Program :

→ code start from

int main () { function }

function → It is an entity which we provide one or multiple input and may be it can give output




```
int main(){

}
```

→ print → "XYZ"

cout << "XYZ" ;

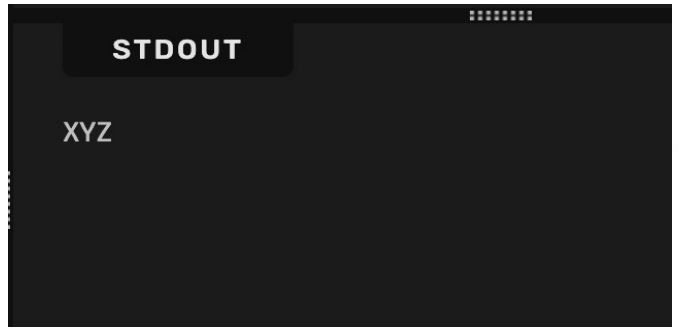
- cout → Print
- << → Syntax
- "XYZ" → String
- ; → end of statement

header file → iostream

namespace → section of code containing definitions

```
SOURCE - main.cpp
1  #include<iostream>
2  using namespace std;
3  int main(){
4      cout << "XYZ" ;
5  }
```

} → Contain whole code



→ Here it print XYZ

#include <iostream> → Preprocessor directive (include all iostream file)

using namespace std → we are using standard namespace definition

int main → Starting of code

cout → used for printing

→ Content between " ____ " is a string as we take XYZ in our code

SOURCE - main.cpp

```cpp
#include<iostream>
using namespace std;
int main(){
    cout << "XYZ" ;
    cout << endl ;
    cout << "ABC" ;
}
```

→ endl is used for new line

STDOUT

```
XYZ
ABC
```

SOURCE - main.cpp

```cpp
#include<iostream>
using namespace std;
int main(){
    cout << "XYZ" << endl;
    cout << endl ;
    cout << '\n';
    cout << "ABC" ;
}
```

→ can also use for new line insted of endl

STDOUT

```
XYZ


ABC
```

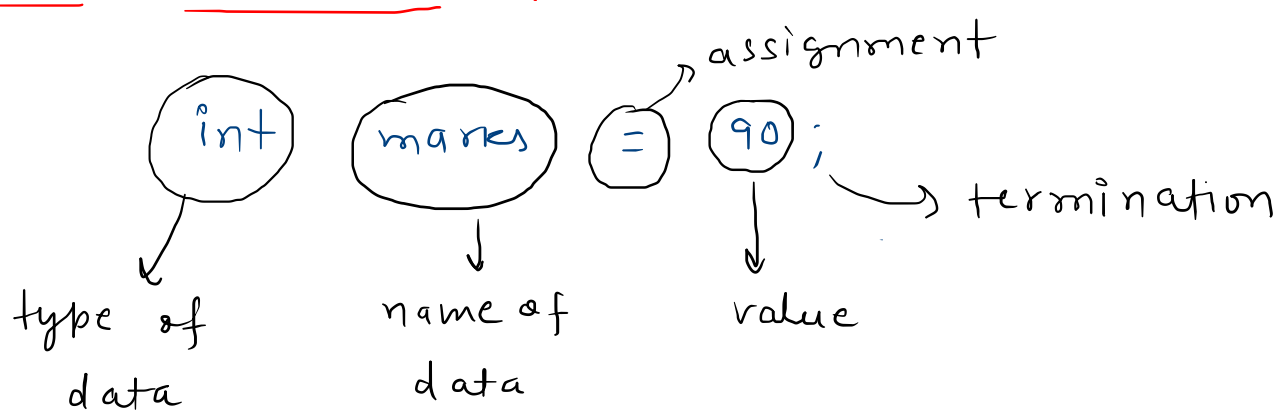
SOURCE - main.cpp

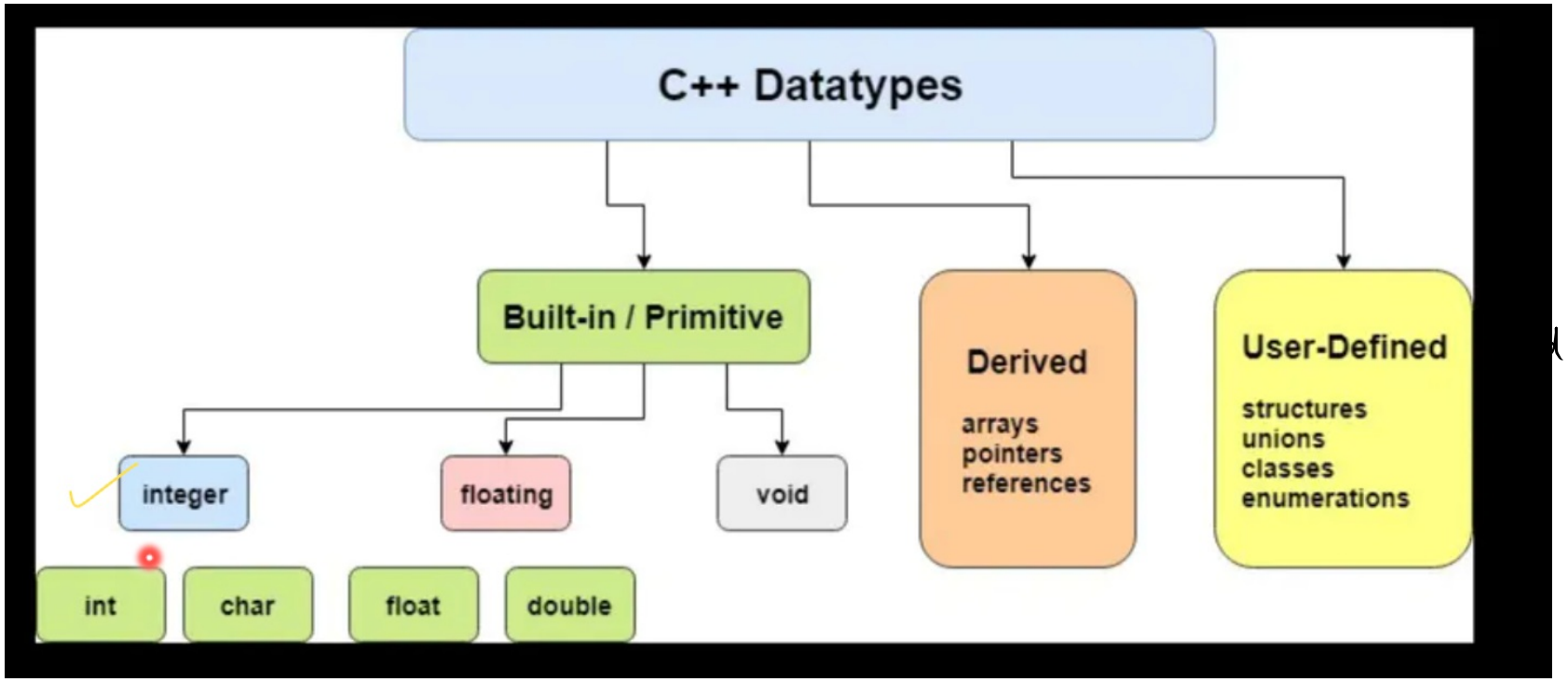
```cpp
#include<iostream>
using namespace std;
int main(){
  // cout << "XYZ" << endl;
  // cout << endl ;
  // cout << '\n';
  // cout << "ABC" ;
}
```

all codes are commented

→ Here // in front of code means we commented the code and it will not execute in our program

Datatypes and variables :





int marks = 90 ;

↓ datatype ↓ variable name ↓ value

→ int a = 5 ; (it makes a block of 4 byte which store value 5)

address = 108

4 byte

→ for char → 1 byte

→ int main → return an integer type value (like return 0)

→ void main → return no value

→ variable → It is a name assigned to a memory location.

→ datatype → type of data stored in variable

→ datatype tell two things :-

→ type of data (int, char, float, double)

→ the memory it takes to store data.

→ int → 2, 4, 9, - - (integer values)

char → A, B, a, b, - - -

float → 2.6, 7.9, - - (decimal values)

double → 2.7987, - - .

SOURCE - main.cpp

```cpp
#include<iostream>
using namespace std;
int main(){
    // int -> 4 byte
    int num = 50;
    cout << num << endl ;
    cout << sizeof(num) << endl;

    // char -> 1 byte
    char c = 'k';
    cout << c << endl;
    cout << sizeof(c) << endl;

    // float -> 4 byte
    float point = 1.69;
    cout << point << endl;
    cout << sizeof(point) << endl;

    // long -> 4 byte for 32 bit system and 8 byte for 64 bit system
    long number = 23;
    cout << number << endl;
    cout << sizeof(number) << endl;
}
```

STDOUT

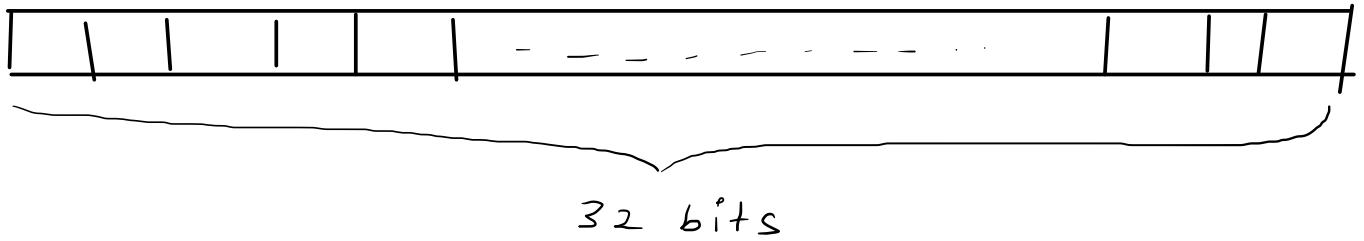
```
50 → num
4 → size of num
k → c
1 → size of c
1.69 → Point
4 → size of Point
23 → number
8 → size of number
```

Naming Conventions rules for Variables are:

- It should begin with an alphabet.
- There may be more than one alphabet, but without any spaces between them.
- Digits may be used but only after alphabet.
- No special symbol can be used except the underscore (_) symbol. When multiple words are needed, an underscore should separate them.
- No keywords or command can be used as a variable name.
- All statements in C++ language are case sensitive. Thus a variable A (in uppercase) is considered different from a variable declared a (in lowercase).

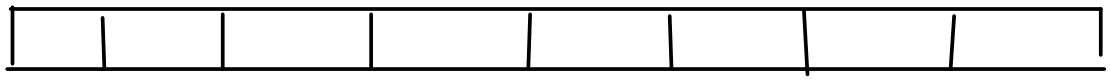
How data is Stored

int → 4 byte → 32 bits



a = 5 (101) → binary of 5

→ char 1 byte (8 bits)



char ch = 'a'; a → ASCII value 97 (store binary of 97 in the boxes)

→ bool flag = True

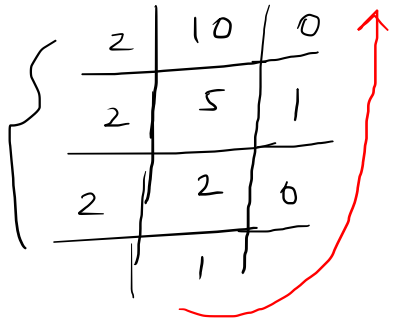
bool datatype store true (1) or false (0)

→ take 1 byte (8 bits)

→ we can store it in 1 bit but still take 8 bit because smallest space we can take is 8 bits i.e 1 byte.

→ binary equivalent of 10

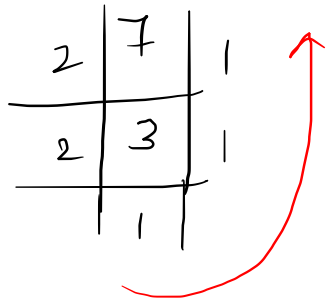
divided by 2 at each stage



2	10	0
2	5	1
2	2	0
	1	

→ 1010 (binary of 10)

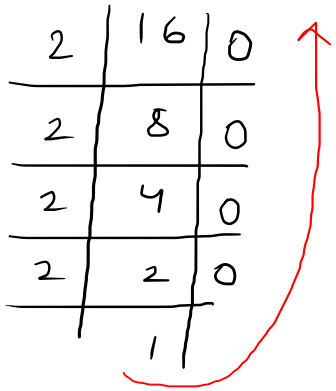
→ 7



2	7	1
2	3	1
	1	

→ 111 (binary of 7)

→ 16



2	16	0
2	8	0
2	4	0
2	2	0
	1	

→ 10000 (binary of 16)

→ For negative numbers

1's compliment →

101 → flip 1 to 0 and 0 to 1

010 → 1's compliment of 101

→ 11010 → 00101

→ 101010 → 010101

→ 111101 → 000010

} all are 1's compliment

→ 2's compliment

Find 1's compliment and then add 1

→ 2's compliment of 101

101

1's compliment → 010

+1 → 010
1

011 (2's compliment)

→ 1011

1's compliment → 0100

+1 → +1

0101

→ 1 0 0 0

1's compliment → 0 1 1 1

+1 → +1

1 0 0 0

(+1 → 2

(10)

→ For negative numbers. ∴

→ ignore -ue sign

→ find binary equivalent

→ take 2's compliment

a = -5

(i) 5

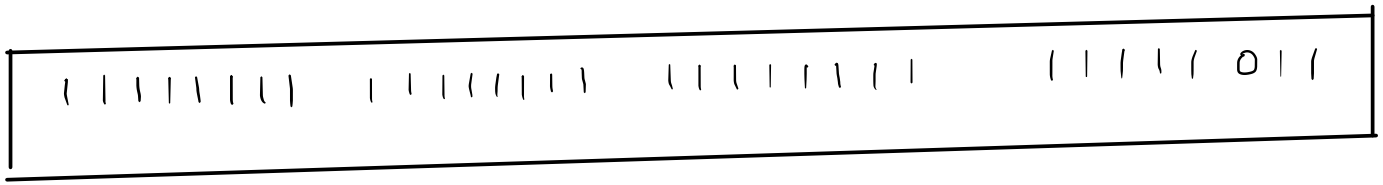
(ii) 00000000 00000000 00000000 00000101

(iii) 2's comp.

1's comp. 11111111 11111111 11111111 11111010

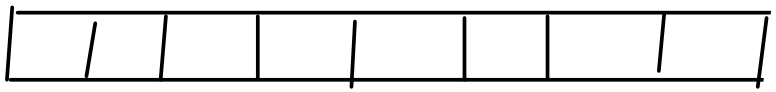
+1

+1

binary of -5 ⟶ 

→ Ranges of data types :-

→ char 1 byte → 8 bits



all can fill up with 1 or 0
So every box have 2 inputs.

Total combination = $2^8 = 256$

range → 0 to 255

0 to $2^8 - 1$

(Generic formula)

[0 to $2^n - 1$]

→ int 4 byte → 32 bits

range → 0 to $2^{32} - 1$

→ long (in 64 bit system) 8 byte → 64 bits

range → 0 to $2^{64}-1$

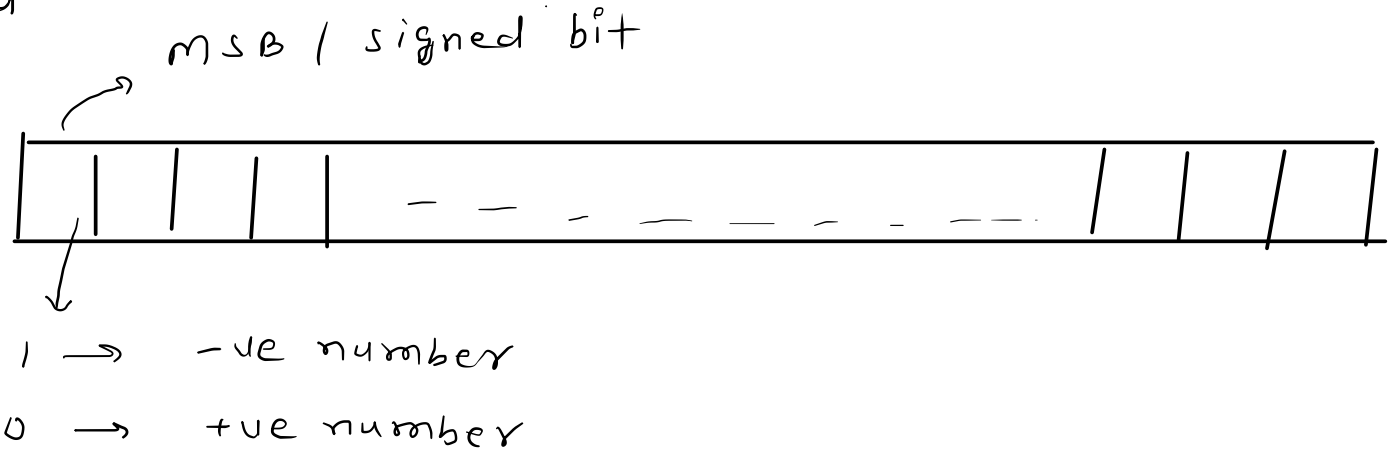
signed vs unsigned data

- by default we create signed data

→ signed can store both +ve and -ve data

→ unsigned only store +ve data

for signed



so now range will be -2^7 to 2^7-1

Generic formula (-2^{n-1} to $2^{n-1}-1$)

⇒ **Operators**

Arithmetic ⟶ + − × / %.

SOURCE - main.cpp

```cpp
#include<iostream>
using namespace std;
int main(){
  int a = 5;
  int b = 10;

  cout << a+b << endl;
  cout << a-b << endl;
  cout << a*b << endl;
  cout << b/a << endl;
  cout << b%a << endl;
}
```

STDOUT

```
15
-5
50
2
0
```

if we do $3/2$ it will give 1 output.

$\frac{int}{int} = int$, $\frac{double}{int} = double$, $\frac{float}{int} = float$

Relational

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

`!=` ↓ not equal to

`==` ↓ equal to

SOURCE - main.cpp

```cpp
#include<iostream>
using namespace std;
int main(){
  int a = 20;
  int b = 10;
  cout << (a>b) << endl;
  cout << (a<b) << endl;
  cout << (a<=b) << endl;
  cout << (a>=b) << endl;
  cout << (a!=b) << endl;
  cout << (a==b) << endl;
}
```

STDOUT

```
1
0
0
1
1
0
```

→ Output 1 for True and 0 for false

Logical

`&&` , `||` , `!`

SOURCE - main.cpp

```cpp
#include<iostream>
using namespace std;
int main(){
    int age = 20;
    int car = 12;
    if(age >= 18 && car >= 1){
        cout << "License Approved" << endl;
    }
}
```

STDOUT

```
License Approved
```

&& is true if both conditions are true

SOURCE - main.cpp

```cpp
#include<iostream>
using namespace std;
int main(){
    int age = 2;
    int car = 12;
    if(age >= 18 || car >= 1){
        cout << "License Approved" << endl;
    }
}
```

STDOUT

```
License Approved
```

|| is true if one of the condition is true

→ Taking input

use cin

cin >> var_name ;

SOURCE - main.cpp

```cpp
#include<iostream>
using namespace std;
int main(){
  int marks;
  cin >> marks;
  cout << "Your mark is" << endl;
  cout << marks;
}
```

STDIN

```
20
```

STDOUT

```
Your mark is
20
```