# Introduction to C++

Lect 1 * **Programming language** :- A programing language is a computer language that is used by programmers to communicate computers.
It is a set of instructions written in any specific language such as Java, C++, C... to perform a specific task.

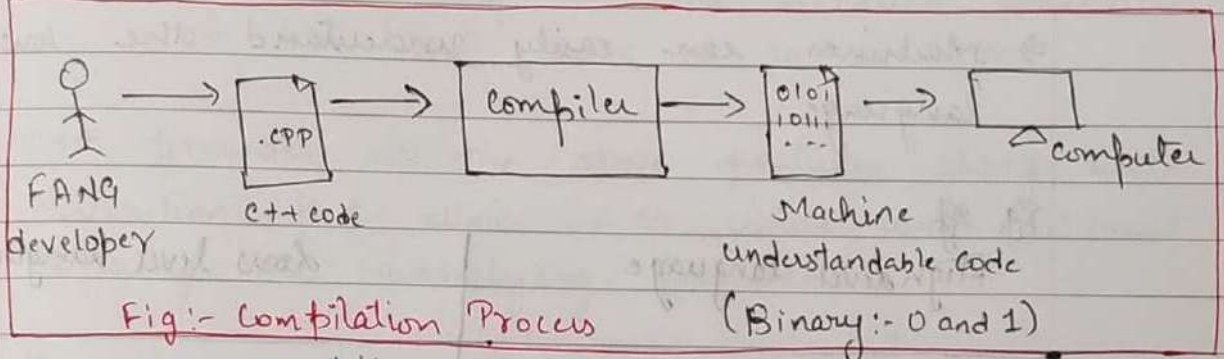



Fig :- Compilation Process

i.e compiler converts code which developers write into binary code which machine understands and hence, in a way the programmers communicate with the computer by the help of a compiler.

⇒ C++ is a high level general purpose programming language

- Object Oriented ①
- Efficient and speed ② } why C++
- Game development ①
- Real time Systems ②
- Software development ③ } where C++ is used

** ③ memory mgmt at low level

⇒ Real life Eg is Adobe has most of its code in C++.

**Homework** :- Is C++ a high level language or a low level language. what is the difference b/w the two?

Ans: Both high level language and low level language are programing language's types. The main difference between them are as follows

⇒ Programmers can easily understand or interpret or compile high level language

whereas

⇒ Machines can easily understand the low level language

| High level language | Low level language |
|---|---|
| → Less memory efficient | → high memory efficient |
| → Easy to understand | → Tough to understand |
| → Debugging is easy | → Debugging is comparatively complex |
| → machine independant | → machine dependant |
| → compiler or interpreter for translation | → assembler for translation |
| → portable | → not portable |
| Eg:- C, C++, Java, Python, etc | Low level language<br>Machine language → 0's and 1's<br>Assembly language → x86 assembly, ARM assembly |

M T W T F S S
Page No.:
Date:
YOUVA

# Is C++ high level language or low level language?

* Features of a high level language
  (i) Use English like words
  (ii) Cannot run directly on computer hardware, requires a translator to translate high level instruction code to machine language code
  (iii) Require coding for the detailed procedure of how the task is to be done.

* C++ provides all the above features along with abstraction that allow us to write code without bothering the underlying hardware.

* Also C++ can be used for direct memory manipulation so it also behaves as a low level language

* Oftenly, refered as a middle level language.

Summary :-

C++ combines the efficiency and control of low level languages with the abstraction and expressiveness of high level languages. Thus, according to Stroustrup, C++ can be both a low level and high level language, depending on how its used and the perspective from which its considered.

Bjarne Stroustrup, the creater of C++, has refered C++ as a "multi-paradigm programming language"

Lect2 Print Namaste Duniya

1. main() is the starting point of execution of the program.

2. `return 0;` means successful execution of the code, if any other number apart from 0 is returned then it indicates that the code didn't execute successfully.

3. the header file iostream contains the std namespace and other functionalities of input and output

4. the std namespace consists the definition of the identifier cout

Thus, To use cout we need both iostream header file as well as the std namespace.

5. cout is used to output | print something

6. With cout, always the insertion operator (<<) is used to print something

7. endl is used to print from next line or to mark the end of the current line

8. ; marks the end of statement in the code

All the above points are the most basic blocks of a C++ code as shown in the below code

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

Homework:- ① alternative of endl.
② explore std::cout
③ what if return -1 ?
④ what are preprocessor directives?

① \n is the alternative of endl.

Eg:-

```
cout << "Namaste Duniya \n";
```

| ② cout | std::cout |
|---|---|
| → using namespace std; must be written in the program | → std::cout can be used if "namespace std" was not declared previously |

Thus, std::cout is just an implicit initialization of the std library performed inside the function i.e alongwith the main computation

③ Returning values such as return 1 or return -1 means the program is returning an error.

④ All the statements starting with a # symbol are known as preprocessor directives.

Preprocessing is a phase among the 7 phase of compilation of a program. The preprocessor can import the content of other program files into the source code file, and expand the macros (which are mostly constant values or expression with a name that be used throughout a program), etc.

Every directive is a single line command containing the following

(i) # symbol at the begining

(ii) a preprocessor instruction after the # symbol
Eg:- #include <iostream>

(iii) arguements based on the type of directive

i.e <iostream> is an arguement for #include

#include <iostream>, #define PI 3.14, etc are examples of preprocesser directives

## Lect 3 Variables and Datatypes

Variable:- A variable is a name given to a memory location so that it can be accessed and manipulated

Datatype:- The datatype specifies the type of info the variable will store associated with its also specifies its size

```
datatype  variable_name = value;
int       age           = 25;
```

↳ This creates a block of memory in the computer (enough) equivalent to the size of an int which has the name age and has the value 25

(4B)

age → variable name
25 → value.

⇒ Thus, the datatype tells us about both the type of data and size of data associated with it

By default, a variable is initialised to a garbage value i.e if a variable is only declared but not initialised, then it stores a garbage value.

**Ques** Why does boolean occupy 1B in memory even though a true or false only need 1b of space ie 0 or 1 as false or true?

**Sol$^{n}$** Boolean variable stores either true or false.

true = 1 = $(00000001)_2$

false = 0 = $(00000000)_2$

But the smallest addressable memory space is 1B i.e 1B can be considered as the smallest memory unit and hence a Boolean also occupies 1B in memory even though required is 1b.

**Home work**

① Conversion from binary to decimal and vice versa.

Decimal to Binary

① Converting 5 into binary.

~~2|5~~
~~2|2~~ ~~1~~

| | | |
|---|---|---|
| 2 | 5 | remainder |
| 2 | 2 – | 1 |
| 2 | 1 – | 0 |
| | 0 – | 1 |

i.e $5 = (101)_2$

② Converting 9 into binary

| | | |
|---|---|---|
| 2 | 9 | remainder |
| 2 | 4 – | 1 |
| 2 | 2 – | 0 |
| 2 | 1 – | 0 |
| | 0 – | 1 |

i.e $10 = (1001)_2$

M T W T F S S

Page No.: YOUVA

Date:

## Binary to decimal

(1) Convert $(101)_2$ into decimal

$\underset{2^2\ 2^1\ 2^0}{101}$

$(101)_2 = (1\times 2^2) + (0\times 2^1) + (1\times 2^0)$

$= (1\times 4) + (0\times 2) + (1\times 1)$

$= 4 + 0 + 1$

$= 5$

i.e $(101)_2 = 5$

(2) Convert $(1001)_2$ into decimal

$\underset{2^3\ 2^2\ 2^1\ 2^0}{1001}$

$(1001)_2 = (1\times 2^3) + (0\times 2^2) + (0\times 2^1) + (1\times 2^0)$

$= (1\times 8) + (0\times 4) + (0\times 2) + (1\times 1)$

$= 8 + 0 + 0 + 1$

$= 9.$

i.e $(1001)_2 = 9.$

Ques How data is stored in memory?

Data is stored in memory only in the form of 0s and 1s i.e binary form because it is machine understandable.

- Positive integers:- 5 will be stored as 101 because it is a positive number so it is directly converted into a binary number

- Negative integers:- -5 will be stored in its 2's complement format because it is a negative integer

$5 = (101)_2$ | 1's (5) = $(010)_2$ | 2's (5) = $(011)$

2's comp = 1's comp + 1.

i.e $-5 = (011)_2$

Ques3 Derieve the formula $-2^{n-1}$ to $2^{n-1}-1$ (signed) & 0 to $2^n-1$ (unsigned)

Ans Let us assume a 3 bit number

| Decimal | unsigned (+ve) | signed (+ve & -ve) (2's complement) | Decimal |
|---|---|---|---|
| 0 | 000 | 000 | 0 |
| 1 | 001 | 001 | 1 |
| 2 | 010 | 010 | 2 |
| 3 | 011 | 011 | 3 |
| 4 | 100 | 100 | -4 |
| 5 | 101 | 101 | -3 |
| 6 | 110 | 110 | -2 |
| 7 | 111 | 111 | -1 |
| 8 | 1000 | | |

↓ 4 digit from here

↳ sign bit :- 0 = positive, 1 = negative

∴ Range = 0 to 7

i.e. = 0 to 8-1

= 0 to $2^3-1$

We assumed 3 bit numbers, so $n=3$.

∴ unsigned range = 0 to $2^n-1$

Range = -4 to 3

= $-(2^2)$ to $(2^2-1)$

= $-(2^2)$ to $(2^2-1)$

= -4 to (4-1)

= -4 to 3

We assumed 3 bit numbers, so $n=3$

∴ signed range is $-2^{n-1}$ to $2^{n-1}-1$

Page No.
Date
YOUVA

Lect 4 User Input in C++.

Input stream - "cin"

Cin is the identifier, that enables us to take user input in a C++ program.

```
#include <iostream>
using namespace std;

int main() {
    1. int age; //declaration of variable.
    2. cout << "Enter your age \n";
    3. cin >> age; //user input of age
    4. cout << "my age is : " << age << endl;
```

Line 1. ⇒ The variable "age" is declared.
Line 2 :⇒ The user is asked for age
Line 3: ⇒ The variable "age" initially has a garbage value which is replaced by the value entered by the user
Line 4: ⇒ Print statement.

NOTE:- while taking boolean as an input, we should only enter 0 or 1 and never use true or false.

Homework:- ① cin.ignore()
② cin.fail()
③ getline(cin, 10)
④ :: operator

① cin.ignore() :-

Lec 5 Control flow statements.

Control flow statements are used in decision making :-

1. If statement
2. If else statement
3. If - else - if statement
4. If - else if - else
5. Nested if.

1 If statement

Syntax :-

```
if (condition)
  {
      //Execute only when condition is true
  }
```

i.e the code within the if block is executed only if when the condition is true. If the condition is false, then the code in the if block is skipped and execution starts from the next line after the if block.

2 If - else block

Syntax :-

```
if (condition){
      //Execute only when cond^n is true
  }
else {
      //Execute when cond^n is false
  }
```

i.e the code within the if block is executed only when the condition is true.

If the condition is false, then only the code present within the else block is executed

> NOTE:- Either the code within the if block is executed or the code within the else block is executed i.e both will never execute together.

After execution of either "if" or "else" depending on whether the condition is true or false respectively, the execution starts from the next line after the "if-else" block

3 If-else if block

Syntax:-

```
if (condition 1) {
    //execute when cond^n 1 is true
}
else-if (condition 2) {
    //execute when cond^n 2 is true
}
else if (condition 3) {
    //execute when cond^n 3 is true
}
:
```

The flow of execution is same as "if-else" but only the block whose condition is true will get executed and then execution starts from the next line after the "if-else if" block.

If no condition is satisfied then none of the blocks are executed and execution directly jumps to the next line after the entire if-else-if block

| NOTE:- Only 1 of all the blocks will be executed |
|---|

4 If-else if-else block

Syntax:-

```
if (condn 1){
    //Execute when condn 1 is true
}
else if (condn 2){
    //Execute when condn 2 is true
}
:
else{
    //Execute when none of the above condn's is true
}
```

The flow occurs same as if-else if block, the only difference is that if no conditions are satisfied then the code within the else block is executed.

5 Nested-if block

```
if (condn 1){
    if (condn 2){
        =
    }
    else{ }
}else{ }
```

Syntax :-

```
if (condn 1) {
    if (condn 2) {
        ;
        =
    }
    else {
        ;
        =
    }
}
else {
    ;
    =
}
```

Nested if means an if statement within another if statement.

Nested if can be used to check multiple conditions

The flow of execution is similar to if-else block, but here ~~the~~ we have an if-else block which gets executed only when the condition of the outer "if" is satisfied.

## Lect 6 Switch Case in C++

The switch statement is also a control flow statement. It is an alternative to using a series of if else statements when we need to compare a variable against multiple values.

```
switch(expression) {
    case value 1:
        // Executed if expression equals value 1
        break;

    case value 2:
        // Executed if expression equals value 2
        break;

    default:
        // Executed if none of the cases match
```

Syntax ↑

⇒ Always remember using the break statement or else all the cases below the satisfied case will also get executed.

⇒ The default case is optional i.e even if it is not included in the code then also the code works fine.

Conditions pertaining to switch

i. Expression type :- We cannot use strings, float or any non integral values in the switch expression. We can use characters, integers, enum, ...

ii. Unique case values :- Always use unique case values i.e two cases can't be same.

iii. No range checking :- Expression must be a constant within a case i.e we can't use case to check range.

```
int age=12;
switch(age){
    case(age<20):
        cout<<"hello";
}
```

This is an invalid case because case(constant) is valid but age<20 isn't a constant

```
int age=12;
switch(age>10){
    case 1:
        cout<<"Hello";
        break;
    case 0:
        cout<<"false";
```

This is a valid case because (age>10) will either be true (1) or false (0).

iv. Fall through behaviour :-

If break statements are not used then all the cases below the satisfied case also gets executed which is known as fall through behaviour.

Homework :- More about cases

Lect 7 Ternary operator in C++

Syntax →

```
condition ? expression_if_true : expression_if_false;
```

```
#include <iostream>
using namespace std;

int main(){
    int age = 12;
    (age > 18) ? cout << "can vote" : cout << "can't vote";
    return 0;
}
```

Output :- can't vote

executed when age>18

executed when age<18

Since age = 12, so "can't vote" is the output.

Lect 8 For and while loop.

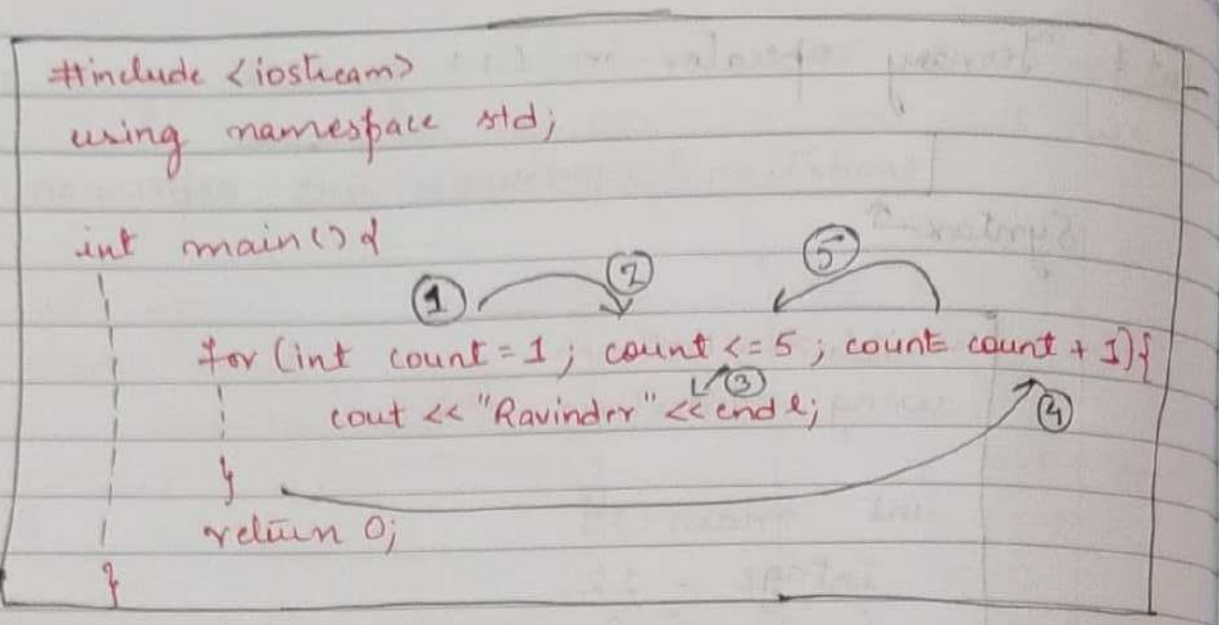
Whenever, we need to execute a block of code multiple times, we can use loops. There are many loops such as for loop, while loop, do while loop etc.

(i) For loop.

Syntax →

```
for(initialization; condition; update){
    // code to be executed in each iteration
}
```

```
#include <iostream>
using namespace std;

int main() {
    for (int count = 1; count <= 5; count= count + 1){
        cout << "Ravinder" << endl;
    }
    return 0;
}
```




:- The above is a for loop which prints "Ravinder" for 5 times.

Step ① :- Initialise a count variable to 1
Initialisation

Step ② :- Check the condition `count <= 5`
condition

Step ③ :- If the condition is true then execute the code within the for loop.
Code to be iterated If the condition is false, exit the for loop

Step ④ :- If the condition was true in step 3 then the code was executed and now the ~~conditions~~ count variable will be updated
updation

Step ⑤ :- ~~Check~~ the condition go to step ② and repeat
Repitition

→ Output:- Ravinder (count = 1; count <= 5; count = count+1)
Ravinder (count = 2; count <= 5; count = count+1)
Ravinder (count = 3; count <= 5; count = count+1)
Ravinder (count = 4; count <= 5; count = count+1)
Ravinder (count = 5; count <= 5; count = count+1)
(count = 6; count <= 5
6 <= 5
not true so here we exit the loop.

## Break Keyword.

The break keyword is used to break out of the loop or i.e once the execution hits the break statement, then the rest of the iterations are not carried out and execution jumps to the next line of code after the for loop.

```
for(int i=0; i<=10; i=i+1){
    cout << i << " ";
    if (i==5){
        break;
    }
}
```

Output:-
0 1 2 3 4 5

When i=5, the cout is executed which prints 5 in the output, and then the if condition is checked and result is true so the break statement is executed and the rest of the iterations are not carried out.

Thus, break statement moves the execution out of the for loop.

## Continue keyword.

The continue keyword is used to skip the current iteration of the loop and move the execution to the next iteration.

```
for(int i=1; i<=5; i++){
    if (i==4){
        continue;
    }
    cout << i<<" ";
}
```

Output :-

1 2 3 5

during the iteration i=4, the if condition is true and hence "continue" is executed which moves the execution to the next iteration and hence 4 is not printed in the output.

(ii) while loop

Syntax

```
initialisation
while (condition){
    /*code to be executed until
    condition is true */

    //updation
}
```

for loop and while loop, both have the same work i.e ~~work~~ function similarly but are totaly different in syntax

```
#include <iostream>
using namespace std;

int main () {
    int count = 1 //initialisation
    while (count <= 5) { //condition
        cout << "Ravinder" << endl;
        count = count + 1; //updation
    }
    return 0;
}
```

Working exactly like for loop

Output:-

| | | | |
|---|---|---|---|
| Ravinder | (count = 1 ; | count <= 5 ; | count = count+1 |
| Ravinder | (count = 2 ; | count <= 5 ; | count = count+1 |
| Ravinder | (count = 3 ; | count <= 5 ; | count = count+1 |
| Ravinder | (count = 4 ; | count <= 5 ; | count = count+1 |
| Ravinder | (count = 5 ; | count <= 5 ; | count = count+1 |
| | (count = 6 ; | count <= 5 ; | count = count+1 |

not satisfied
so out of loop

Homework:-
1. 1 to 100 counting
2. 100 to 1 counting
3. Name - 50 times
4. 0 to -10 counting
5. 7 table
6. A - Z
7. a - z

(1)
```
for (int i = 1, i <= 100; i = i+1) {
    cout << i << " ";
}
```

(2)
```
for (int i=100; i>=1; i=i-1) {
        cout << i << " ";
}
```

(3)
```
for (int i=0; i<50; i++) {
        cout << "Ravinder" << endl;
}
```

(4)
```
for (int i=0; i>=-10; i--) {
        cout << i << " ";
}
```

(5)
```
for (int i=0; i<=10; i++) {
        cout << "7 * " << i << " = " << 7*i << "\n";
}
```

(6)
```
for (char ch='a'; ch<='z'; ch++) {
        cout << ch << " ";
}
```

(7)
```
for (char ch='A'; ch<='Z'; ch++) {
        cout << ch << " ";
}
```

(8) Are initialization, condition and updation all necessary in the for loop?

Answer `for (initialisation; test_condition; updation);`

Any of these three conditions can be ommitted and still the loop works fine

① for(;;) is an infinite loop equivalent to while(1) or while(true) because there is no test condition.

for(int i=0; ; i++) is also an infinite loop because there is no test condition

for(int i=0; i<=10; ) is also an infinite loop as i will never be updated because there is no updation statement so i=0 is never incremented or decremented

```
int i;
for( ; i<=10; i++){
cout<<i<<" ";
}
```

is also a valid loop and produces the output

0 1 2 3 4 5 6 7 8 9

Even though i is never initialized it takes a default value of 0.

We saw all the cases i.e

① omitting all 3
② omitting only updation
③ omitting only check condition
④ omitting initialisation only

} infinite loop.

To end infinite loop as in case ① ② and ③ break(); or exit(0) can be used.

Lect 9 Do-while and Nested loop.

(iii) do-while loop

Syntax →

```
initialisation
do{
    //code and updation
} while(condition)
```

The do while loop also has initialisation, updation and check condition just like for loop and while loop.

NOTE:- The main point of do while loop is that the code is executed atleast once while using do while loop

⇒ The first iteration is always executed irrespective of whether the condition is satisfied or not which is the main difference between while loop and a do while loop.

```
int i=21;      //initialisation
do{
    cout<<i<<" ";
    i=i+1  //updation
} while(i<=5); //condition
```

→ Output
21
(i=22, i<=5) ✗
↑
false so out of loop

i.e in the first iteration condition is never checked and hence the code produces output ⇒ 21

# Nested Loops

```
for (int i = 1; i<=3; i = i+1) {
    for (int j = 1; j<=3; j = j+1) {
        cout << "i : " << i << ", j : " << j << endl;
    }
}
```

Output :-

```
i : 1, j : 1
i : 1, j : 2
i : 1, j : 3
i : 2, j : 1
i : 2, j : 2
i : 2, j : 3
i : 3, j : 1
i : 3, j : 2
i : 3, j : 3
```

(i=1; i<=3) (j=1; j<=3)
(i=1; i<=3) (j=2; j<=3)
(i=1; i<=3) (j=3; j<=3)
(i=2; i<=3) (j=1; j<=3)
(i=2; i<=3) (j=2; j<=3)
(i=2; i<=3) (j=3; j<=3)
(i=3; i<=3) (j=1; j<=3)
(i=3; i<=3) (j=2; j<=3)
(i=3; i<=3) (j=3; j<=3)

(i=2; i<=3) → true
(j=4; j<=3) → false
(i=3; i<=3) → true
(j=4; j<=3) → false
(i=4; i<=3) → false
(j=4; j<=3) → false

i.e for every iteration of the outer loop the inner loop executes 3 times.

or

for i = 1 ⇒ j = 1, j = 2, j = 3 then
for i = 2 ⇒ j = 1; j = 2, j = 3 and
for i = 3 ⇒ j = 1; j = 2, j = 3

i.e for each value of i, j is executed from j = 1 to j = 3

Homework :-

(1) How does for(int i = 0; i<=5; i++); { // code } work?

All three steps of the for loop carry out as it should be i.e the initialisation of the variable i, updation of i after each iteration and condition check after each iteration, hence in our case the loop runs for i=0, i=1, i=2, i=3, i=4 and i=5 but the body of the loop is empty so it doesn't perform anything. The semicolon at the end of the for statement marks the end of the statement and hence according to the compiler, the body of the loop is empty.

The compiler regards the portion of code

```
{
    cout << "hello ji"
}
```

as a seperate block of code and executes it independantly of the for loop.

(2)
```
int main() {
    if (cout << "hello ji") {
        cout << "kaise ho saare";
    }
    return 0;
}
```

Sol ⇒ cout << "hello ji" statement outputs the string "hello ji" to the standard output stream

⇒ The << operator returns a reference to the cout object

⇒ The if statement evaluates the result of the output operation. In C++, any non zero value

is considered true, while a zero value is considered false

⇒ Here, the cout << "hello ji" operation is successful, it returns a reference to the cout object, which is implicitely converted to true. Thus, the cond" inside the if statement is evaluated to true, and the code inside the if block is executed.

Output :- hello ji Kaise ho saare!

(3)
```
int main() {
    int i;
    if (cin >> i) {
        cout << "hello ji";
    }
    return 0;
}
```

Sol"

The cin >> i statement tries to read input from the standard input stream into the variable i. If the input operation succeeds (i.e if a value is entered and assigned to i), the condition inside the if statement evaluates to true, and the code inside the if block is executed.

If the input operation fails (if the input is not a valid integer), the condition inside the if statement ~~fails~~ evaluates to false, and the code inside the if block is skipped.

Lec 10 Operators in C++.

An operator is a symbol that operates on a value to perform specific mathematical or logical computations. They form the foundation of any programming language.

1) Unary operator

i) Pre increment & Post Increment
(++a) (a++)

⇒ Pre-increment ⇒ ① Increase first
② use the increased value

⇒ Post-increment ⇒ ① Use the value first
② Increase after using.

```
int a = 5;
cout << ++a;
```

Output :- 6
(increment then use)

```
int a = 5
cout << a++;
```

Output :- 5
(use then increment).

However, in both the case the final value of a changes from 5 to 6.

ii) Pre-decrement & Post decrement

⇒ Pre-decrement ⇒ ① Decrease first
② Use the decreased value

⇒ Post decrement :⇒ ① Use the value first
② Decrement ~~the~~ after using.

```
int a = 5;
cout << --a;
```

Output :- 4

```
int a=5
cout << a--;
```

Output :- 5

In both cases, the final value of a = 4. after the code is executed

2) Binary Operators → Arithematic operators
→ Relational operators
→ Logical operators
→ Bitwise operators
→ Assignment operators.

① Arithematic operators :-

① Addition :- a+b
② Subtraction :- a-b
③ Multiplication :- ~~a~~ a*b
④ Division :- a/b
⑤ Modulus :- a % b

```
int a=10, b=5;
cout << a+b <<endl;
cout << a-b <<endl;
cout << a*b <<endl;
cout << a/b <<endl;
cout << a%b <<endl;
```

Output :- 15 (add)
5 (sub)
50 (mul)
2 (divide)
0 (remainder)

The modulus operator is a very resource consuming operation and hence is not advised to use until no other alternative is there.

Note :- ① int / int = int
② float / int = float
③ double / int = double

(ii) Relational operator.

Always returns only a true or false value ie 1 or 0 respectively.

① >
② <
③ >=
④ <=
⑤ == (Equal to)
⑥ != (not equal to)

Eg :-
```
cout << (10>5) << endl;
```
→ Output :- 1

(iii) Logical operator → && (and)
→ || (or)
→ ! (not)

These are used to combine multiple conditions

→ && :- all the conditions should be true, even if one condition is false; then output is false

→ || :- if any of the condition is true, then the output is true i.e atleast one condition should be true for output to be true

→ ! :- the not operator negates the output
i.e !(true) = false
!(false) = true

&& operation

| condn 1 | condn 2 | Output |
|---|---|---|
| true | false | false |
| true | true | true |
| false | true | false |
| false | false | false |

→ True only when all the conditions are true
or
False if atleast one condition is false.

|| operation

| condn 1 | condn 2 | Output |
|---|---|---|
| false | true | true |
| false | false | false |
| true | true | true |
| ~~false~~ true | false | true |

→ false only when all the conditions are false.
or
True if atleast one condition is true.

! operation

| Input | Operator | Output |
|---|---|---|
| true | !(true) | false |
| false | !(false) | true |

(iv) Assignment Operator

:) = → int a=5; i.e to assign value.
:) += → a=a+1; <=> a+=1; are the same
:) -= → a=a-1; <=> a-=1; are the same
:) *= → a=a*5; <=> a*=5; are the same
:) /= → a=a/2; <=> a/=2; are the same
:) %= → a=a%10; <=> a%=10; are the same

→ short hand notation

(v) Bitwise operator :- (&, |, ~, <<, >>, ^).

:) & (Bitwise 'and).

| a | b | a & b |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

Works exactly like && but here the AND operation is carried out on each bit individually.

Eg $5 = (0101)_2$
$3 = (0011)_2$

```
5 & 3 ⇒  0 1 0 1   (5)
       & 0 0 1 1   (3)
       -----------
         0 0 0 1
```

Now, $(1001)_2 = 9$ Thus, $5 \& 3 = 9$

:) | (Bitwise or)

| a | b | a \| b |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

Works exactly like || but bitwise.

Eg :- $5 = (0101)_2$
$3 = (0011)_2$

```
5 | 3 ⇒  0 1 0 1   (5)
       | 0 0 1 1   (3)
       -----------
         0 1 1 1
```

Now, $(0111)_2 = 7$ Thus, $5 | 3 = 7$

## ⇒ Tilda operator (~)

| a | ~a |
|---|---|
| 0 | 1 |
| 1 | 0 |

i.e it performs the bitwise not operation on each bit independently.

$5 = (0101)_2$ and $3 = (0011)_2$

$\sim 5 = (1010)_2$ $\sim 3 = (1100)$

The MSB is 1 in both case and so they are negative numbers evaluated using 2's complement:

$\sim 5 = (1010)$ $\sim 3 = (1100)$

$\sim 5 = -6$ $\sim 3 = -4$

Thus, $\boxed{\sim 5 = -6 \text{ and } \sim 3 = -4}$

NOTE:- The above calculations are done assuming 4 bit numbers. However, actually numbers in CPU may be 16 bit or 32 bit long.

## ⇒ Left Shift operator (<<)

Shifts all the bits by one position to left and suffix it by an additional 0

Eg:- 5<<1

(0101)$_2$ << 1

Now, $5 = (0101)_2$
after we perform left shift

0101 changes to 1010
↳ addition of 0 as a suffix

Now, $(1010)_2 = 10$

Thus, left shift multiplies a number by 2! when performed once
If left shift is performed twice then the number is multiplied by $2^2 = 4$
If left shift is performed thrice, then the number is multiplied by $2^3 = 8$.

Eg:- $5 << 1 \Rightarrow (1010)_2 = 10$ $(8+2)$ or $5 \times 2$
$5 << 2 \Rightarrow (10100)_2 = 20$ $(16+4)$ or $5 \times 4$
$5 << 3 \Rightarrow (101000)_2 = 40$ $(32+8)$ or $5 \times 8$

## ⇒ Right Shift (>>)

0101
10100
101000

Works same as left shift but the bits are shifted in the right direction as the name suggests right shift.

Eg:- $5 >> 1 \Rightarrow (0101)_2 = 5$ and after right shift
$(0010)_2 = 2$
Wkt, $5/2 = 2$

i.e in a way, the right shift operator is working like it divides the operand by 2 each time it is performed.

## ⇒ XOR operator (^)

| a | b | a^b |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

→ Whenever both input are different output is 1
or
whenever both input are same output is 0

**Note :- a^a = 0**

Now, $5 = (0101)_2$ and $3 = (0011)_2$

then 5^3 =
```
  0101
  0011
  ----
  0110   = 6
```

i.e 5^3 = 6

Homework :- ① Which datatypes can we apply bitwise operators

② Can we use ~ on floats?

① Bitwise operators can be applied to

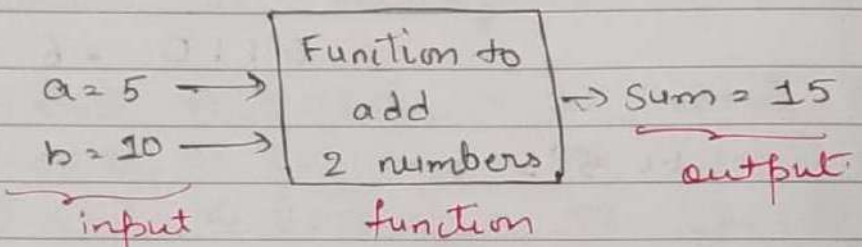
int, unsigned int, short, unsigned short, long, unsigned long, char, unsigned char and enums

Bitwise operators are not defined for float, double or long double and neither for boolean. Other produces compilation error if used with bitwise operators

② Using modulo operator with floating point operands (float and double) is not allowed in C++. They will give compilation error. Instead fmod() from <cmath> can be used.

## Lec 11 Functions in C++

A function is a way to group code into a single unit. It takes inputs, processes them and return a result. Functions helps in organising code, making it more readable and maintainable.




Function anatomy:-

```
return_type  function_name (input parameters){
        //function body
}
```

⇒ The return type is the datatype ~~value~~ of the value which the function is going to return.

⇒ If the function does not return anything, then the return type is void.

```
int sum (int a, int b){
    return (a+b);
}
```

↳ return type = int
function name = sum
input parameters = a and b.

```
#include <iostream>
using namespace std;

int sum (int a, int b){
    int totalSum = a+b;
    return totalSum;
}

int main(){
    int ans = sum(5, 10);
    cout << ans << endl;
    return 0;
}
```

→ Output :-
15

Step 1 :- ans = sum(5, 10)

Step 2 :- the input parameters in the function sum are initialised as a=5 & b=10

Step 3 :- totalSum = 15 (5+10)

Step 4 :- ans = 15 as returned by the function sum

Step 5 :- the output is is printed.

Just as we have shown in the code above, the definition of the function used by the main() function is written before the main function so that the compiler can find the function sum() while executing the main() function

However, if we declare the function first then we may swap the order of the two function main() and sum() as ~~shown~~ shown below.

```
#include<iostream>
using namespace std;

void printMyName(); // function declration

int main() {
    printMyName(); // function call
    return 0;
}

void printMyName() { // function definition

    cout << "Ravinder" << endl;
}
```

Output:-

Ravinder

Imp+ Thus, either define the called function before the calling function or declare the called function before the calling function and swap their order. Best practise is to define the called function before the calling function

Lect 12 # Binary & decimal number systems

## ⇒ Decimal System

⇒ The decimal number system has the base 10
⇒ It uses digits from 0 to 9
⇒ Base is the number of symbols (digits) a number system uses.

## ⇒ Binary System

⇒ The number system uses the base 2.
⇒ It uses only two digits i.e 0 or 1.

⇒ CPU and memory uses only binary system. ie only 0 and 1 to store and compute everything However, the output on the screen is always in a decimal base.

## ⇒ Decimal to Binary Conversion

### i. Division method :—

1 Divide the number by 2

2 Store the remainder
3 Repeat step 1 and 2 with the quotient until quotient is less than 2.
4 Reverse the bits so obtained.

Eg:- Convert 10 into binary :-

$(10) = (\ ?\ )_2$

| Division | Remainder |
|---|---|
| $10/2 = 5$ | 0 |
| $5/2 = 2$ | 1 |
| $2/2 = 1$ | 0 |
| $1/2 = 0$ | 1 |

Thus $10 = (1010)_2$

ii Bitwise Method :- (imp)

1 Obtain bit with bitwise AND operation
2 Right shift N by 1 (N = N>>1)
3 Repeat above steps till N>0
4 Reverse the bits so obtained.

NOTE:- 10 will be stored as 1010 in memory as discussed earlier so we can perform 10 & 1 . & The compiler will interpret it as (1010) & (0001) because all the computations and storage inside a CPU happens in binary only

(1) 1010
0001
0000 = 0

(2) 101
001
001 = 1

(3) 10
01
00 = 0

(4) 1
1
1 = 1

Now, on reversing we get 1010.

Thus, $10 = (1010)_2$

Bitwise method is better than division method, because everything is computed bitwise inside the CPU.

```
#include <iostream>
#include <cmath>
using namespace std;
int decimalToBinary (int n){
    int bit = 0;
    int i = 0;
    int binaryNo = 0;
    while (n>0){
        bit = (n & 1);
        binaryNo = bit * pow(10, i++) + binaryNo;
        n = n >> 1;
    }
    return binaryNo;
}
int main (){
    int n;
    cin >> n;
    cout << decimalToBinary(n);
    return 0;
}
```

Output

7

111.

## ⇒ Binary to Decimal Conversion

1 Multiply each digit by its place value
2 ~~Ad~~ Add up all place values
3 Sum is the decimal number.

$(1010)_2 = (?)_{10}$

$= (1 \times 2^3) + (0 \times 2^2) + (1 \times 2^1) + (0 \times 2^0)$
$= (8) + (0) + (2) + (0)$
$= 8 + 2$
$= 10$ i.e $(1010)_2 = (10)_{10}$

## Lect 13 Type Casting in C++

⇒ Allows to change the data type of a variable from one type to another

⇒ Crucial when you need to perform operations involving variables of different data types and the compiler ~~converts~~ promotes one type to a larger type to maintain precision.

int to float

Eg:-
```
int num1 = 10;
float num2 = 5.5;
float result = num1 + num2;
cout << result << endl;
```
→ Output:- 15.5

float is a larger datatype as compared to int so definitely has more precision and hence in this case the result is a float because num 1 is also converted to float from int.

=> float to int.

```
int num1 = 10;
float num2 = 5.5;
int result = num1 + num2;
cout << result << endl;
```

→ Output :- 15

Here just like in previous example the addition of an int and a float will return a float because the compiler implicitely ~~retur~~ converts the result into a float because of high precision.

Since we have specified the datatype of the result as an int ~~so~~ which does not store any precision after point so the decimal point is truncated and the final result is an integer.

=> char to int

```
char ch = 'A';
int a = ch + 1;
cout << a << endl;
```

↳ Output = 66

'A' has the ASCII code 65. We performed ch + 1 and stored it in the variable a and so a stores the value 65 + 1 = 66

=> int to char

```
int a = 97;
char ch = a;
cout << ch << endl;
```

→ Output :- a.

# Explicit type casting in C++.

⇒ Allows to explicitely specify the desired data type during an assignment or operation

⇒ We use the casting operator, which is represented by parantheses containing the target datatype

```
int num1 = 10;
float num2 = 5.5;
float result = num1 + (int) num2;
cout << result << endl;
```

→ Output = 15

Explicit type casting

⇒ double to int

```
double pi = 3.14159265;
int intpi = (int)pi;
cout << intpi << endl;
```

→ Output = 3

Since double got converted into int so its floating point precision is lost and the final answer is an integer i.e 3.

⇒ float to char

```
float no. = 65.35;
char charValue = (char)no;
cout << charValue << endl;
```

→ Output = 'A'

float to char, the precision is truncated and 'A' has ASCII 65

NOTE :-

1. (int)/(int) = int
2. (int)/(float) = float
3. (float)/(int) = float

If we do not want to change the datatype of the operands from int but still want result in float then we can use explicit type casting.

```
int a=10;
int b=3;
float c= a/b;
cout<< c<<endl
```

→ Output :- 3

```
int a=10;
int b=3;
float c= a/(float)b);
cout<<c<<endl;
```

→ Output :- 3.33333

# Live Class 1

Live class with Lakshay Sir.

## Data Types in C++

- Primary
  - Integer
  - Character
  - Boolean
  - Floating point
  - Double
  - void
  - wide character
- Derived
  - Function
  - Array
  - Pointer
  - References
- User defined
  - class
  - structure
  - union
  - Enum
  - Typedef

## cin.ignore()

```
#include <iostream>
int main () {
    char first, last;
    std::cout<<"Please enter your first name followed
                by your last name \n";

    first = std::cin.get(); //one char i/p only
    std::cin.ignore (256, ' ');

    last = std::cin.get();

    std::cout<<"Your initials are "<<first<<last<<"\n";

    return 0;
}
```

| Input :- | Output :- |
| --- | --- |
| Ravindr Tetarwal | Your initials are RT |

When we wrote Ravinder Tetarwal in the console, it got stored in the input buffer.

Cin takes input from the input buffer

cin.get() ← takes ip only one character

cin.ignore(256,' '); ← ignores 256 characters ~~until~~ unless a space comes.
i.e space acts as a delimeter here.

Thus, first stores 'R' and last stores 'T' from the input buffer and hence output is RT

| | Data Type | Size (Byte) |
| --- | --- | --- |
| 1 | bool | 1 |
| 2 | char | 1 |
| 3 | unsigned char | 1 |
| 4 | short | 2 |
| 5 | unsigned short | 2 |
| 6 | int | 4 |
| 7 | unsigned int | 4 |
| 8 | long | 4 or 8 |
| 9 | unsigned long | 4 or 8 |
| 10 | long long | 8 |
| 11 | unsigned long long | 8 |

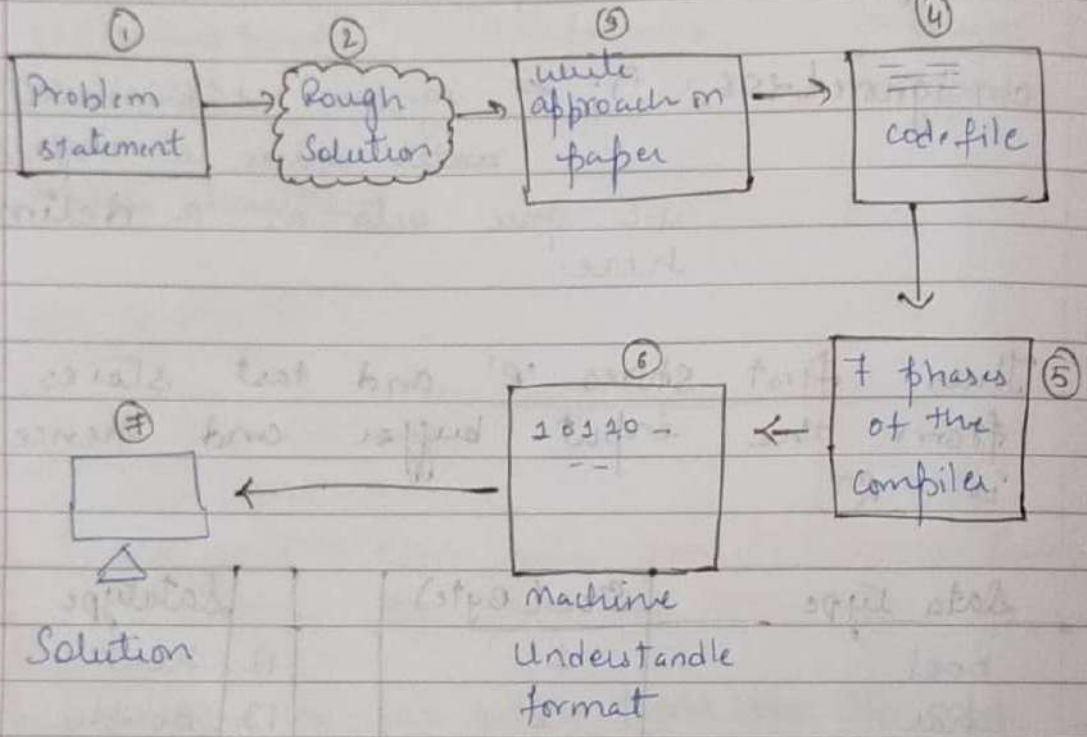
| | Datatype | Size |
| --- | --- | --- |
| 12 | float | 4 |
| 13 | double | 8 |
| 14 | long double | 12 or 16 |
| 15 | w_chat | 2 or 4 |

Live class 2

## Introduction to the programming world.

⇒ How to approach a problem

① ⇒ Understand the problem
② ⇒ Analyse the problem → check i/p value, constraints, requirements
③ ⇒ Logic building / approach / algorithm

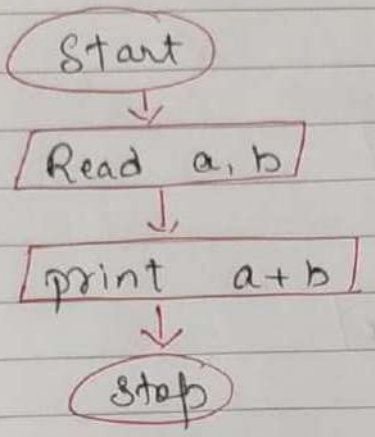



The above 7 steps can be thought as an algorithm of solving problems

### Flow chart and its components:-

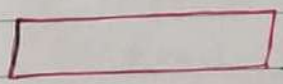
A flow chart is a diagramatic representation that illustrates the sequence of operations

to be performed to arrive at a solution for the problem. It uses various symbols such as arrows, rectangles and diamonds to represent different types of action or steps in a process.




← This is an example of a simple flow chart.

(1) Start & Stop are the terminators and represent the begining and ending of the ~~code~~ flowchart. They are always oval like shaped.

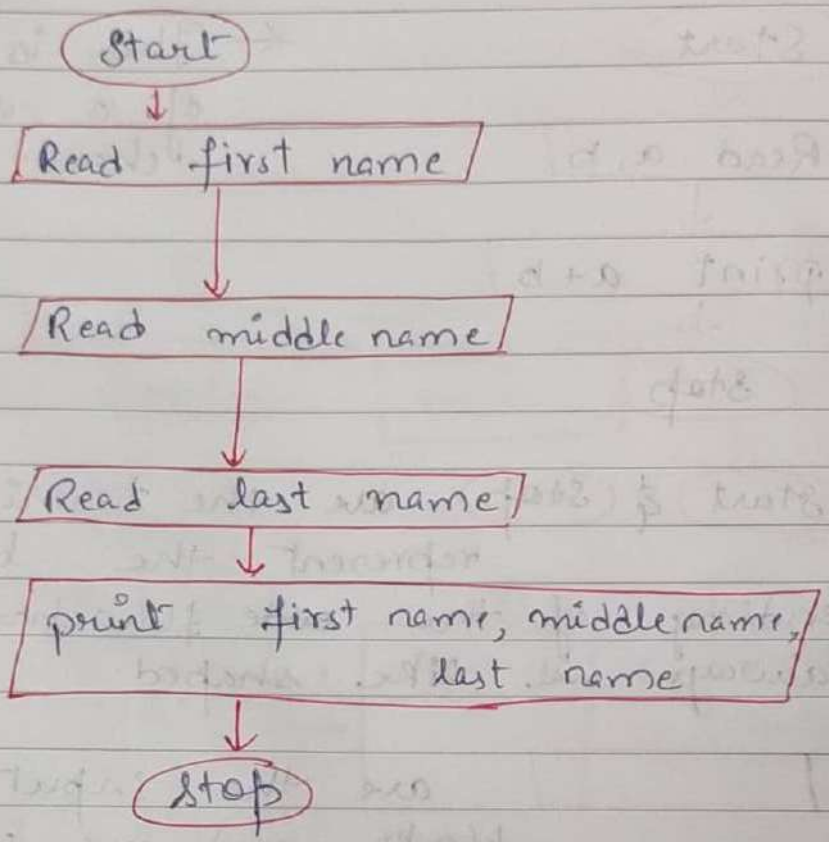
(2)  are the input and output blocks and are used to show what i/p is to be taken from the user and what o/p is to be displayed on the console. They are always parallelogram

(3)  are the process blocks and they ~~mean~~ show processes such as addition, subtraction, etc. They are always a rectangle

(4)  are decision blocks and have a condn within them which is either true or false

(5) ↓ represents the direction of the flow of execution of the code.

Eg:- flow chart to read and print full name of a person



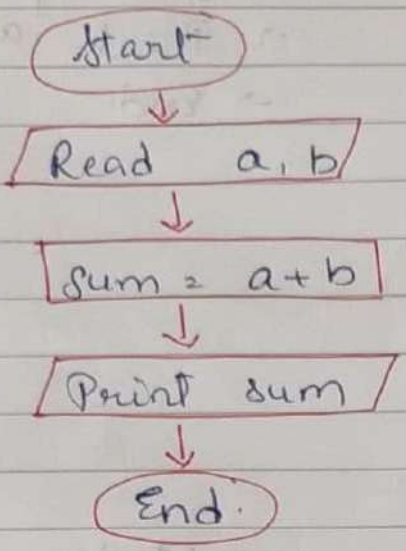

Pseudo code

→ Start
→ Take input first name
→ Take input middle name
→ Take input last name
→ Print firstname, middlename and lastname
→ End/stop

Pseudo code is just another representation of the

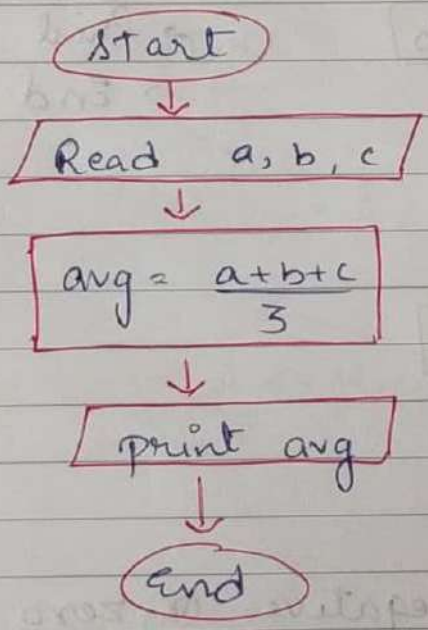
algorithm just like flowchart is a representation of an algorithm.

(1) Print sum of a and b.



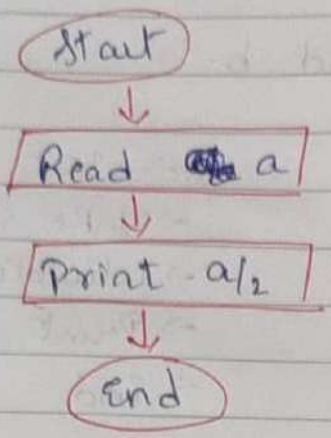

→ Start
→ Take input a and b
→ Calculate sum = a+b
→ Print sum
→ End.

(2) Print average of a, b and c.



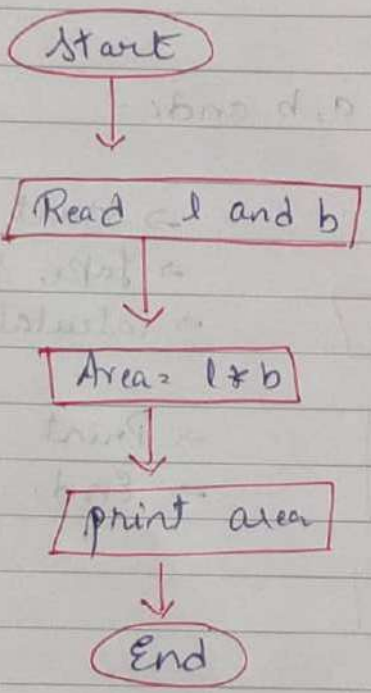

→ Start
→ Take input for a, b & c
→ Calculate avg, $\frac{a+b+c}{3}$
→ Print avg
→ End.

③ Print half of a




→ Start
→ Take input for a
→ Print a/2
→ End.

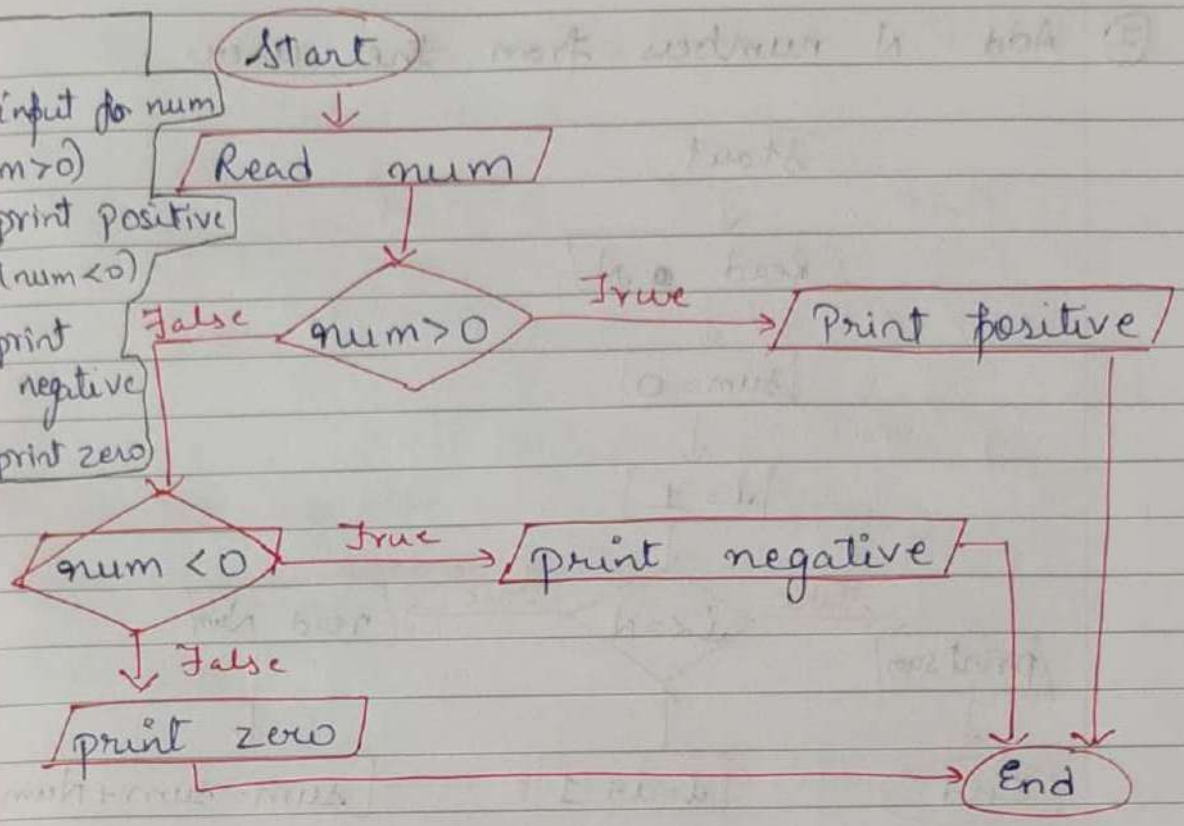
④ Area of a rectangle




→ Start
→ Take input of l and b
→ Calculate area = l * b
→ Print area
→ End.

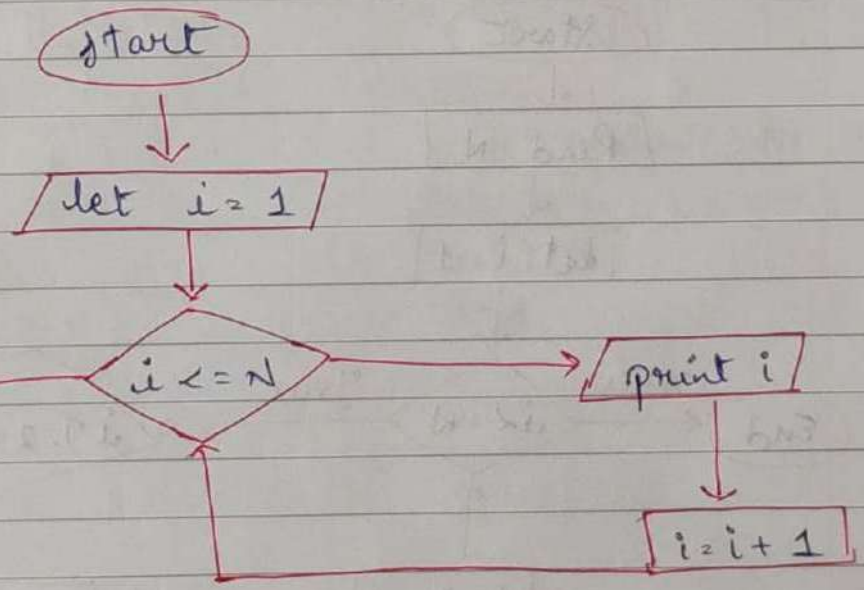
⑤ check positive, negative or zero.

→ Start
→ Take input for num
→ if (num>0)
print positive
→ else if (num<0)
print negative
→ else
print zero
→ End



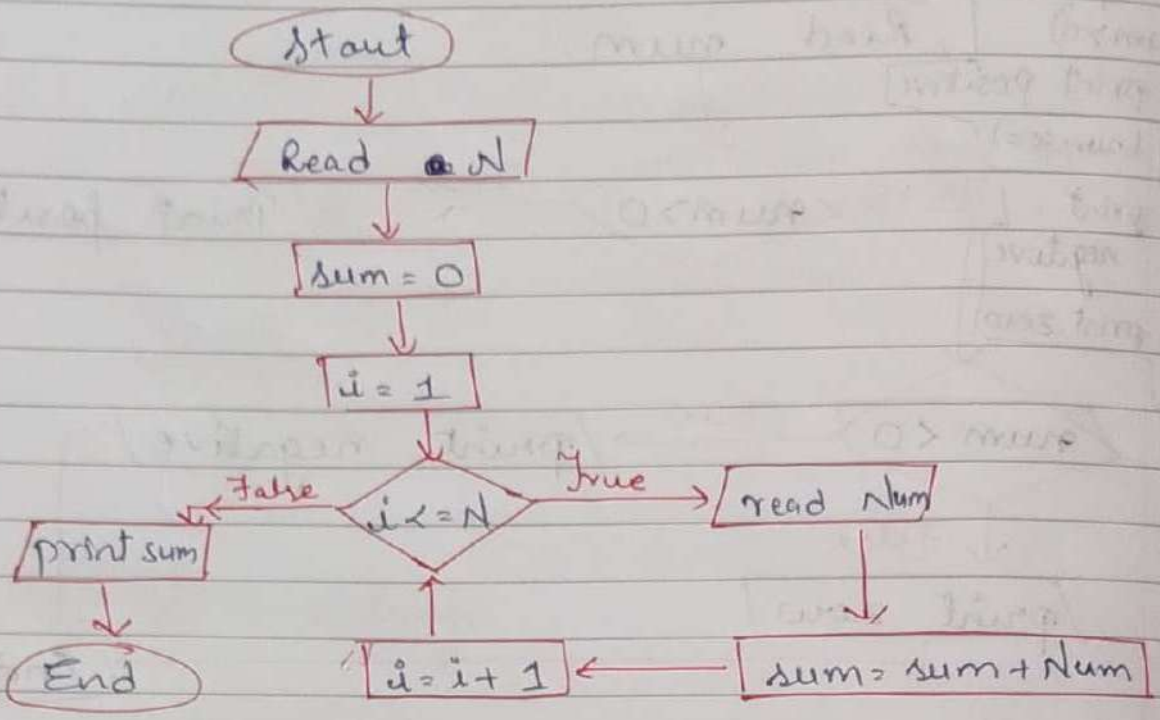

⑥ Print Counting from 1 to N

Ⓐ Start
Ⓑ Take i/p for N
Ⓒ let i = 1
Ⓓ if i<=N
Ⓔ print i
Ⓕ i = i+1
End
Ⓖ goto Ⓓ
else
End.

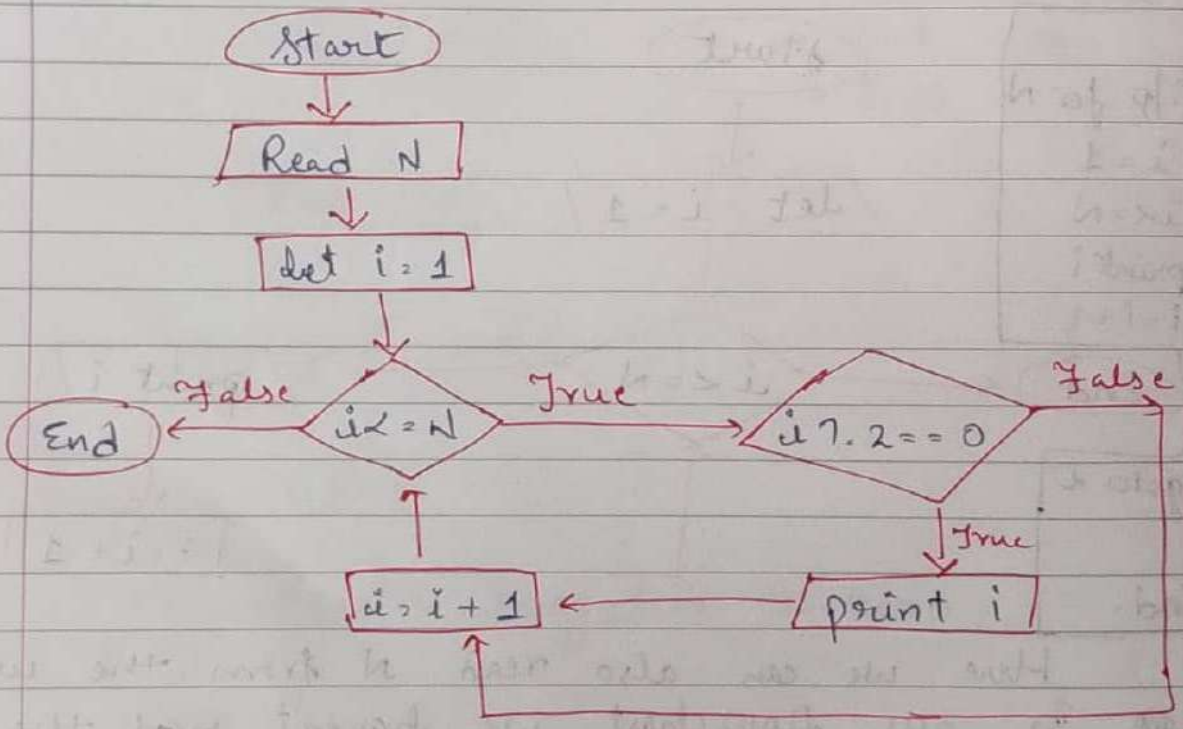



Here we can also read N from the user.
In our flowchart, we haven't read the value of N from the user.

⑦ Add N numbers from the user.



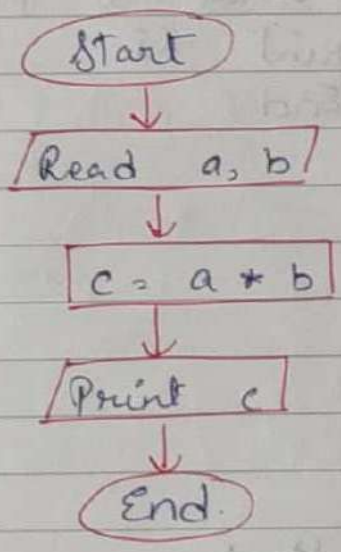

⑧ Print only even numbers from 1 to N



(A) → Start
(B) → Take input for N
(C) → Let i = 1
(D) → if (i <= N)
(E) | | if (i % 2 == 0)
(F) | | | print i
(G) | | | i = i + 1
(H) | | | repeat D
(I) | | else
| | goto G.
(J) → else
End.

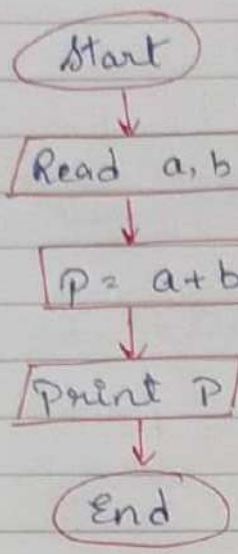
Homework

(1) Multiply two numbers after taking input.



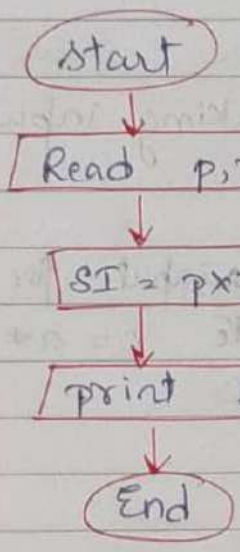

→ Start
→ Read input for a, b
→ Calculate c = a*b
→ Print C
→ End.

② Perimeter of a Triangle



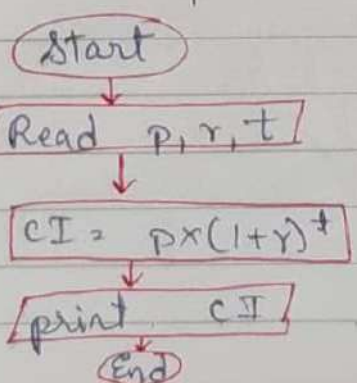

→ Start
→ Take input for a, b and c
→ Calculate P = a+b+c
→ Print P
→ End.

③ Find Simple Interest




→ Start
→ Take input for P, r and t
→ Calculate SI = p×r×t
→ Print SI
→ End.

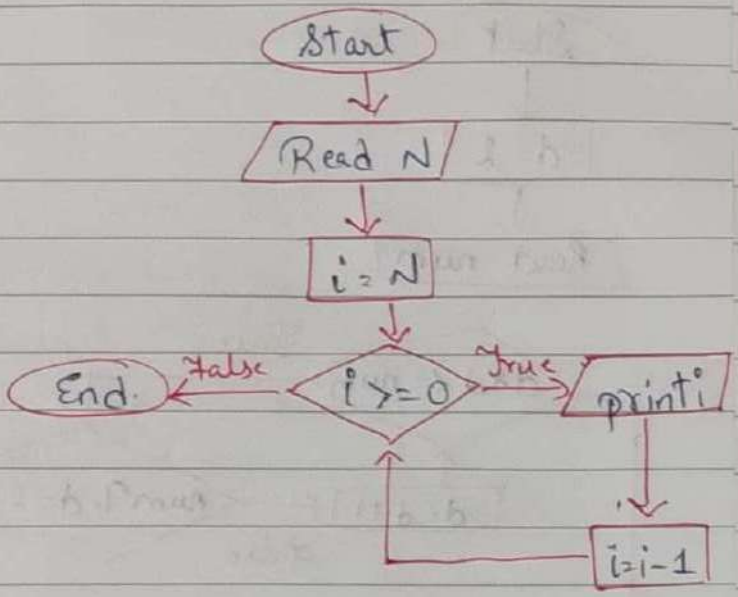
④ Find compound Intrest




→ Start
→ Take input for P, r and t
→ Calculate CI = p×(1+r)^t
→ Print CI
→ End.

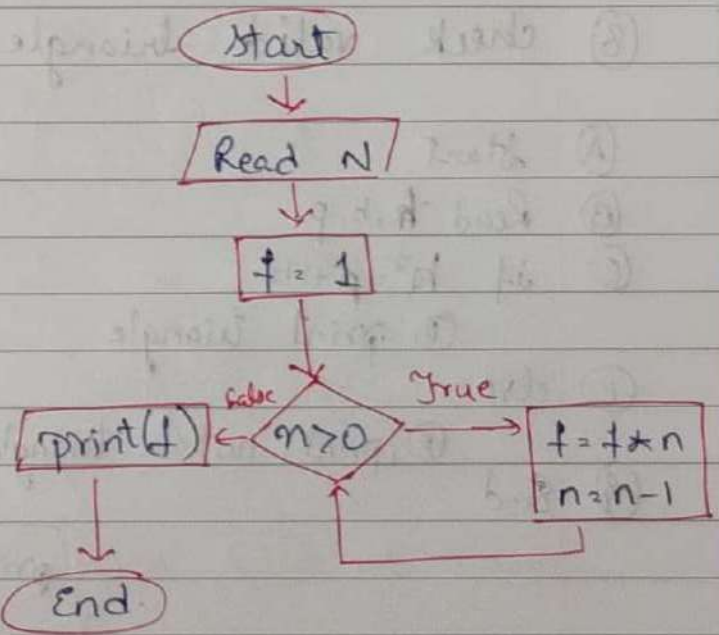
(5) Print counting from N to 1.

A → Start
B → Take input for N
C → Let i = N
D → if (i >= 0)
E  (E) print i
   (F) i = i - 1
   (G) goto D
H → else
    End.

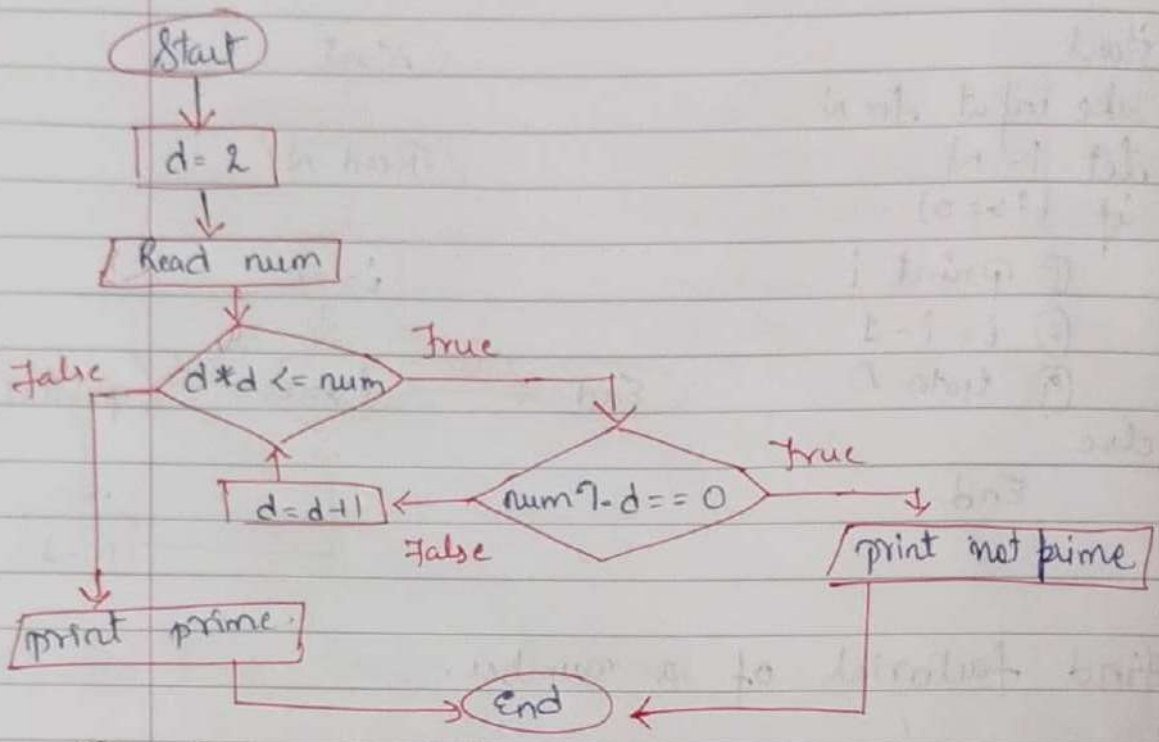



(6) Find factorial of a number.

(A) Start
(B) Take input for n
(C) Let f = 1
(D) if (n > 0)
    (E) f = f * n
    (F) n = n - 1
    (G) goto (D)
(H) else
    print f.
(I) End.

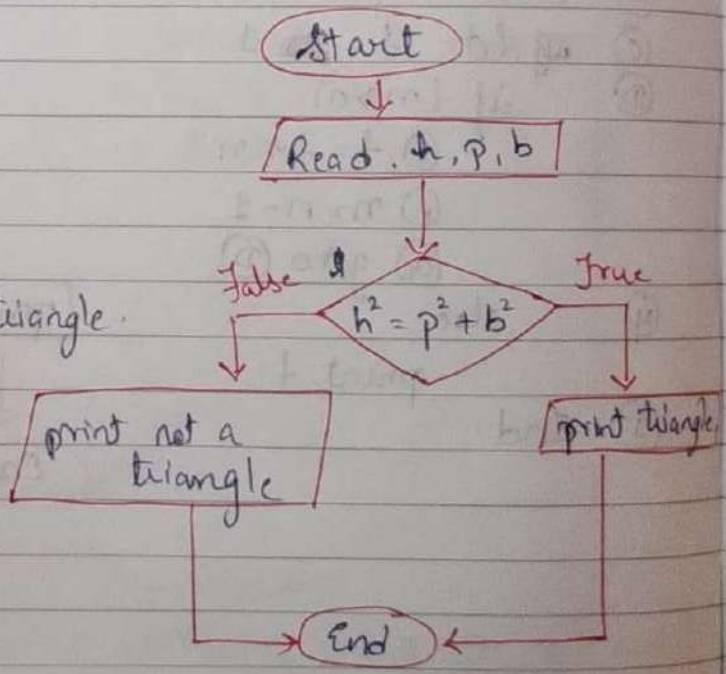


(7) Check if a number is prime or not.




(8) check valid triangle or not

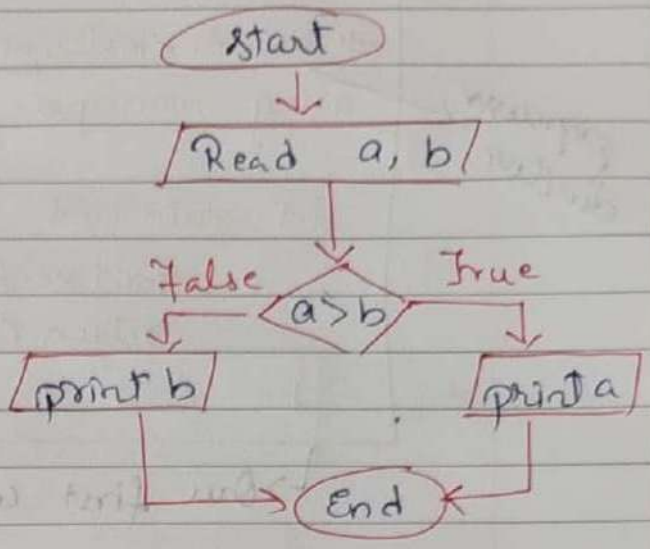
(A) Start
(B) Read h, b, p
(C) if $h^2 = p^2 + b^2$
(D) print triangle
(E) else
(F) print not a triangle.
(G) End




Here h: hypotenuse
p: perpendicular
b: base

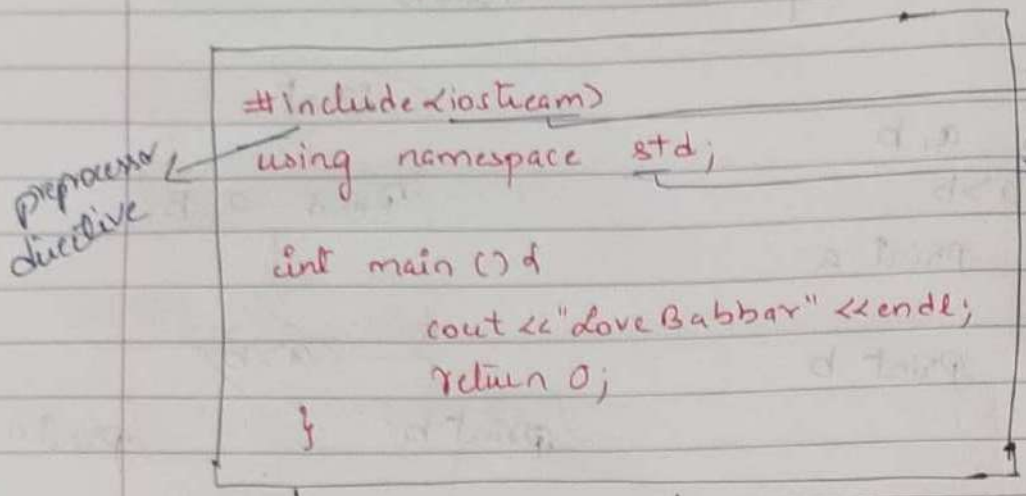
② Print max of two numbers

```
Start
Read a, b
if a>b
    print a
else
    print b
End
```



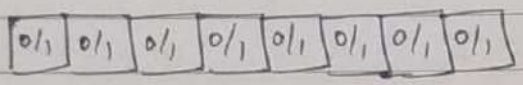
# Live class 3

## Basics of C++ and first program




↳ Our first code (In detail explanation already in previous notes).

Consider a char, we know that a character is of 1B i.e 8 bits and each bit contains 0 or 1

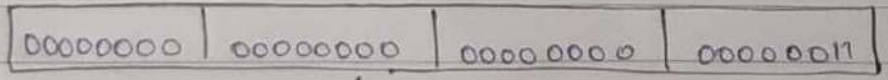



we have are total $2^8 = 256$

Hence, ASCII table contains 256 characters corresponding to 256 permutations

How to find range of any data type

Consider an unsigned integer, where unsigned integer occupies 4B of memory space. 4B = 32 bits and each bit may contain 0 or 1

Eg:- 


↳ 3 stored in memory

Total combinations possible = $2^{32}$ [2 choice for each bit).

∴ Range of unsigned int = 0 to $2^{32}-1$

↓ means 0 or greater than 0 i.e excluding negative numbers.

However, if we considered int datatype which includes all the positive, negative and zero, the range would change.

Range of ~~signed~~ int = $-2^{31}$ to $2^{31}-1$

Formula:- Unsigned range :- 0 to $2^n-1$:
signed range :- $-2^{n-1}$ to $2^{n-1}-1$

↳ Derivation in previous notes.

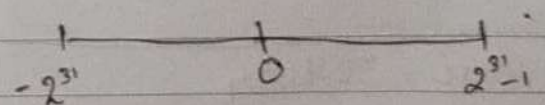
~~Size~~

We can get the size of the datatype using sizeof() operator.

Derivation unsigned :-


($2^{32}$ combination for 32 bits)

signed :-




There are total $2^{32}$ combinations and now

Since negative numbers are included so the positive range is divided by 2.

$\frac{2^{32}}{2} = 2^{32-1} = 2^{31}$ Thus, higher limit = $2^{31}-1$

The rest $2^{31}$ numbers are the negative numbers

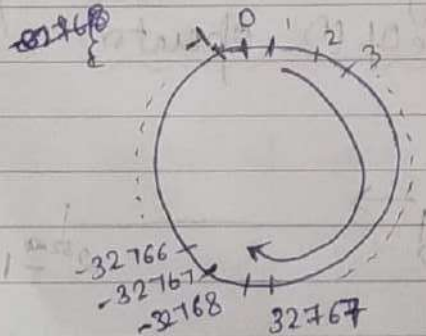
So: **unsigned range = $-2^{31}$ to $2^{31}-1$**

## Cyclic property

We know that short is of 2B = 16 bits.

$\therefore$ Range = $-2^{15}$ to $2^{15}-1$
= $-32768$ to $(32768-1)$
= $-32768$ to $32767$

Thus, the short datatype behaves as follows




Thus the name cyclic property.

```
short value = 32768;
cout << "Value is : " << value << "\n";
```

↳ Output :- Value is : -32768

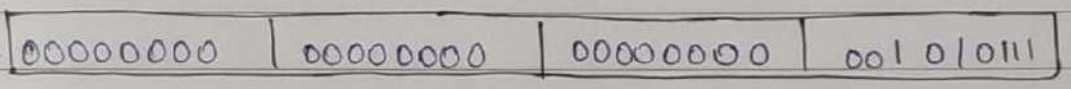
Similaly, when value = 32769 in the code, then the output will be -32767.
and so on.

Variable naming convention:-

1. Should begin with an alphabet
2. There may be more than 1 alphabet but without any spaces between them
3. Digits may be used but only after alphabet.
4. No special symbol apart from an underscore (_) which may seperate multiple words
5. No keywords or commands as variable name
6. All statements in C++ are case sensitive

Brain Teaser

Consider a memory block storing 65 as follows




↳ Binary representation of 65.

How does the compiler know if it is a char 'A' or integer?

This info is carried is by the datatype and if the the datatype is an int then all the 4B are returned, else if the datatype is a char which is only 1B in size then only the last byte is returned i.e 00101011 by default.

Live class 3.

Doubt class with Lakshay Sir.

## Namespace

⇒ Namespace is used for scoping.
⇒ The scope resolution opeator needs to be used to specify a particular namespace

```
#include<iostream>
using namespace std;

namespace Lakshay{
        void driving(){
            cout<<"Hello! Lakshay is driving";
            cout<<"\n";
        }
        void dessParty(){
            cout<<"Welcome to Lakshay's party";
            cout<<"\n";
        }
}

namespace Ravinder{
        void driving(){
            cout<<"Hello! Ravinder is driving";
            cout<<"\n";
```

```
void letsParty(){
        cout<<"Welcome to Ravinder's party";
        cout<<"\n";
}

int main(){
        Ravinder::driving();
        Ravinder::letsParty();
        Lakshay::driving();
        Lakshay::letsParty();
        return 0;
}
```

(::) ← is the scope ~~resolution~~ resolution operator and it specifies the scope of the element used as seen in the above example

## Prime Numbers

Prime numbers are numbers that are only divisible by 1 or itself

```
#include <iostream>
using namespace std;

bool isPrime(int n){
    if(n<=1){
        return false;
    }
    for(int i=2; i<n; i++){
        if(n%i==0){ return false; }
    }
    return true;
}
```

```
int main(){
    int n;
    cout<<"Enter the value of n"<<n";
    cin>>n;
    for (int i=1; i<=n; i++){
        if (isPrime(i)){
            cout<<i<<" is prime\n";
        }
        else{
            cout<<i<<" is not prime\n";
        }
    }
return 0;
```

↳ Output:- Enter the value of n

15

1 is not prime
2 is prime
3 is prime
4 is not prime
5 is prime
6 is not prime
7 is prime
8 is not prime
9 is not prime
10 is not prime
11 is prime
12 is not prime
13 is prime
14 is not prime
15 is not prime

cin.fail() (not so imp)

↳ checks if input from user was failed or not.

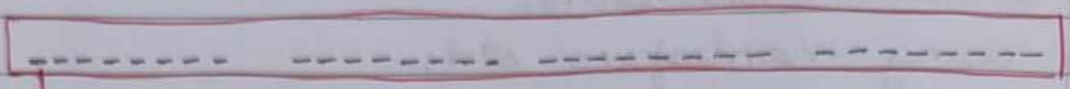
```
#include <iostream>
using namespace std;

int main(){
    int num;
    cout<<"Enter an integer"<<endl;
    cin>>num;
    if(cin.fail()){
        cout<<"failed"<<endl;
    }
    else{
        cout<<"success"<<endl;
    }
    return 0;
}
```

↳ Output:- 56
success
Lakshay
Failed.

Live class 4

# Operators, conditionals and loops

How +ve/-ve numbers are stored?



MSB ← Most significant bit.

The MSB decides whether the number is a positive number or a negative number.

MSB :- 0 : +ve number
1 : -ve number

1's complement :- flip all the bits.

Eg :- 00000000 00000000 00000000 00000101

↳ $(5)_2$

1's $(5)_2$ 11111111 11111111 11111111 11111010

2's complement :- Add 1 to 1's complement

Eg :- 2's (5) = 11111111 11111111 11111111 11111010
+ 1

2's (5) ⇒ 11111111 11111111 11111111 11111011

Positive numbers are stored directly in its binary form in the memory

+5 would be stored as 00000000 00000000 00000000 00000101

↑

MSB = 0, so +5

Negative numbers :-

① → ignore the negative sign
② → store its 2's complement.

-5 would be stored as

11111111 11111111 11111111 11111011 <=> 2's(5).

↑

MSB = 1 so -5

Verification :-

Consider memory contains

11111111 11111111 11111111 11111011

① MSB = 1 so definitely it is a negative number

② We take 2's complement for negative numbers

Thus, 00000000 0000 0000 0000 0000 00000 100 ← 1's

(1's) +1 ← 0000 0000 0000 0000 0000 0000 00000101 ← 2's

↳ represents 5.

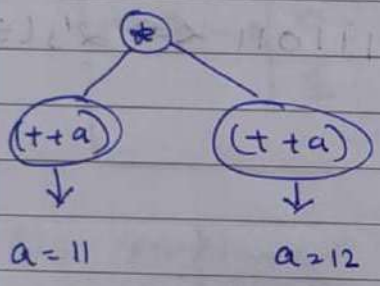
Thus, this verifies that it was -5.

## Ambiguity in code

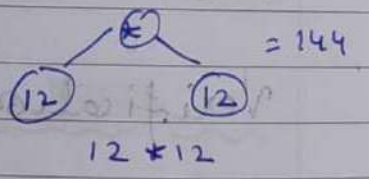
```
int a = 10;
cout << (++a) * (++a);
```

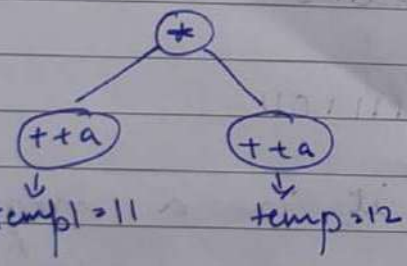
→ Output :- The output of such code is compiler dependant. It may give output as 132 or 144 on different compilers.

Output ⇒ 132 and 144 explanations

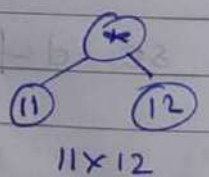



i.e finally a=12 so we get




i.e finally we get




Thus, such code are ambigous code and may produce different output on different compilers.

Similarly,

```
(a++)* (++a)
(++a)* (a++)
(a++)* (a++)
```

← They are all ambigous operations

NOTE:- Never use such ambigous code because the output is not fixed and varies on different compilers

Live class 5

Let's Solve Patterns

```
#include <iostream>
using namespace std;

int main() {
    for(int i=1; i<=3; i++) { //Outer loop
        for(int j=1; j<=2; j++) {

            cout << "i = " << i << endl;

            for (int j=1; j<=2; j++) { //inner loop
                cout << "j = " << j << endl;
            }
        }
        return 0;
    }
```

↳ Output :-

→ i=1
  j=1
  j=2
→ i=2
  j=1
  j=2
→ i=3
  j=1
  j=2

i.e if we see carefully, for every value of i we get all the values of j.

We can see that for each value of i we have printed j=1, j=2 i.e inner loop completes all iterations for each i.

NOTE:- i.e for each iteration of the outer loop, the inner loop also completes all its iterations

Outer loop ki every value ke liye inner loop poora chlta hai

```
for (int i=1; i<=3 ; i++) {
        cout << i<< " ->";
        for (int j= 1; j<= 2; j++) {
                cout <<"*";
        }
        cout <<endl;
}
```

→ Output :-

```
1 -> **
2 -> **
3 -> **
```

i.e for each value of i, ** is printed.

Patterns :-

(1) Divide into rows and columns.
(2) number of rows and use in outer loop
(3) number of columns and use in inner loop.

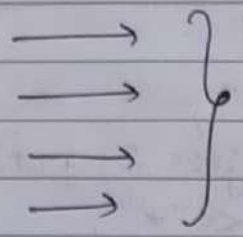
Eg :-

```
      col0 col1 col2 col3
Row0-> *    *    *    *
Row1-> *    *    *    *
Row2-> *    *    *    *
Row3-> *    *    *    *
```

In each row print 4 stars (no. of columns = 4).

It is similar to for each value i print all value of j. Thus, we are using rows in outer loop and column in inner loop; to print 4 stars in each row.

We are printing row wise because the console always gives output from left to right and top to bottom. i.e




① Square Pattern

⇒ Always start the loop from i=0 & j=0

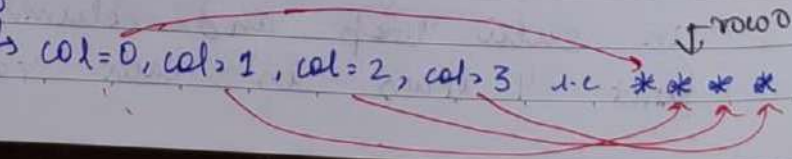
```
int main(){
    for(int row=0; row<4; row++){
        for(int col=0; col<4; col++){
            cout<<"* ";
        }
        cout<<endl;
    }
    return 0;
}
```

→ Output:-

```
* * * *
* * * *
* * * *
* * * *
```

for row = 0

↳ col=0, col=1, col=2, col=3 i.e * * * *



for row = 1
↳ col = 0, col = 1, col = 2, col = 3 i.e * * * * ← row 0
* * * * ← row 1

for row = 2
↳ col = 0, col = 1, col = 2, col = 3 i.e * * * * ← row 0
* * * * ← row 1
* * * * ← row 2

for row = 3
↳ col = 0, col = 1, col = 2, col = 3 i.e * * * * row 0
* * * * row 1
* * * * row 2
* * * * row 3

Thus, for each row (outer loop) the 4 stars (col = 4 → inner loop) are printed and the final output is obtained.

② Rectangle Pattern

col 0 col 1 col 2 col 3 col 4

row 0 → ☆ ☆ ☆ ☆ ☆
row 1 → ☆ ☆ ☆ ☆ ☆
row 2 → ☆ ☆ ☆ ☆ ☆

i.e for each row 5 stars are printed

thus, 3 rows
5 columns

```
int main () {
    for (int row = 0; row < 3; row++) {
        for (int col = 0; col < 5; col++) {
            cout << " * ";
        }
        cout << endl;
    }
    return 0;
}
```

for row = 0
↳ col=0 col=1 col=2 col=3 col=4 5<5 ✗
for row = 1 ★ ★ ★ ★ ★
↳ col=0 col=1 col=2 col=3 col=4 5<5 ✗
★ ★ ★ ★ ★
for row = 2
↳ col=0 col=1 col=2 col=3 col=4 5<5 ✗
★ ★ ★ ★ ★

3<3 ✗

Thus, Output = ★ ★ ★ ★ ★
★ ★ ★ ★ ★
★ ★ ★ ★ ★

In the above examples, the patterns are hard coded but we can take user input on the length and breadth of the pattern.

(3) Hollow Rectangle

Eg:- row 0 ★ ★ ★ ★ ★ (col 0 col 1 col 2 col 3 col 4)
row 1 ★ ★ → length = 4
row 2 ★ ★ → column = 5
row 3 ★ ★ ★ ★ ★
i → (row = 0 to length-1
col = 0 to breadth-1

Observations:- 4 rows
row 0 → 5 stars (first row)
row 1 → 1 star 3 space 1 star
row 2 → 1 star 3 space 1 star
row 3 → 5 star (last row)
(length-1 = 4-1 = 3)

By sym me into

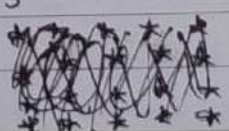
i.e first and last row have 5 stars all the rest rows have stars only in 1st and last column, the rest column have spaces.

```
int main(){
    int length, breadth;
    cout << "Enter the length\n";
    cin >> length
    cout << " Enter the breadth\n";
    cin >> breadth;
    for(int row=0; row<length; row++){
        for(int col=0; col<breadth; col++){
            if(row==0 || row== length-1){
                cout << " * ";
            }
            else{
                if(col==0 || col==breadth-1){
                    cout << " * ";
                }
                else{
                    cout << "   ";
                }
            }
        }
        cout << endl;
    } return 0;
}
```

Output:- Enter the length
4
Enter the breadth
5

```
* * * * *
*       *
*       *
* * * * *
```

By taking symmetrical input, we can convert it into a square (hollow)



## ④ Hollow Square

```
int main (){
    int n;
    cout << "Enter the side of square\n";
    cin >> n;
    for(int row=0; row<n; row++){
        for(int col=0; col<n; col++){
            if(row==0||col==0||row==n-1||col==n-1){
                cout << " * ";
            }
            else{
                cout << "   ";
            }
        }
        cout << endl;
    } return 0;
}
```

i.e when row=0 or col=0 or last row or last col
first row first col row=n-1 col=n-1

then, we will print *, other wise space

Output:- Enter the side of square

5

col=0 col=4 = n-1 = 5-1 = 4

row=0 → * * * * *

* * *

* * *

* * *

row=4 → * * * * *

= n-1 = 5-1 = 4

i.e (row==0 || col==0 || row==n-1 || col==n-1) ⇒

## ⑥ Half Pyramid

```
*
* *
* * *
* * * *
```

```
int main(){
    int n;
    cout<<"Enter the number of rows\n";
    cin>>n;
    for(int row=0; row<n; row++){
        for(int col=0; col<row+1; col++){
            cout<<" * ";
        }
        cout<<endl;
    }
    return 0;
}
```

Output:- Enter the number of rows
5

```
*                ← row=0 ; col<1   ∴ 1 star
* *              ← row=1 ; col<2   ∴ 2 star
* * *            ← row=2 ; col<3   ∴ 3 star
* * * *          ← row=3 ; col<4   ∴ 4 star
* * * * *        ← row=4 ; col<5   ∴ 5 star
```

for row=0, col=0, (col<1) → 1 star
for row=1, col=0, col=1 (col<2) → 2 star
for row=2, col=0, col=1, col=2 (col<3) → 3 star
for r=3, col=0, col=1, col=2, col=3 (col<4) → 4 star
for r=4, col=0, col=1, col=2, col=3, col=4 (col<5)
for r=5, col=0, col=1, col=2, col=3, col=4, col=5 (col<6)

⑥ Inverted half pyramid

```
* * * * *
* * * *
* * *
* *
*
```

```
int main () {
    int n;
    cout << " Enter the number of rows in";
    cin >> n;
    for (int row=0; row<n; row++) {
        for (int col=0; col<n-row; col++) {
            cout << " * ";
        }
        cout << endl;
    }
    return 0;
}
```

→ Output:- Enter the number of rows

```
5
* * * * *
* * * *
* * *
* *
*
```

n = 5

for row=0 (n-row) = (5-0=5)
↳ col=0, col=1, col=2, col=3, col=4 (5<5) ✗

(n-row)=5-1=4 for row=1 → col=0, col=1, col=2, col=3 (4<4) ✗

(n-row)=5-2=3 for row=2 → col=0, col=1, col=2 (3<3) ✗

(n-row)=5-3=2 for row=3 → col=0, col=1 (2<2) ✗

(n-row)=5-4=1 for row=4 → col=0 (1<1) ✗

for row=5 (5<5) ✗

Thus, the output is

```
* * * * *
 * * * *
  * * *
   * *
    *
```

⑦ Hollow half pyramid.

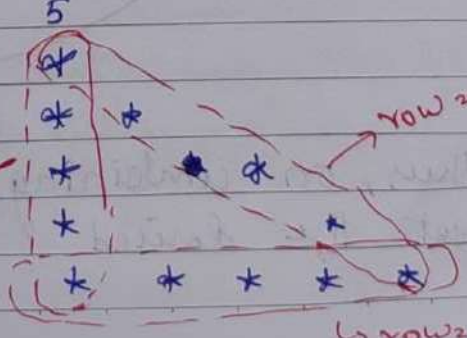
```
int main () {
    cout << " Enter the number of rows \n";
    int n;
    cin >> n;
    for (int row = 0; row < n; row++) {
        for (int col = 0; col < row + 1; col++) {
            if (col == 0 || row == col || row == n-1) {
                cout << " * ";
            }
            else {
                cout << "   ";
            }
        }
        cout << endl;
    }
    return 0;
}
```

Output:-

```
*
* *
*  *
*    *
* * * * *
```

↳ Output:- Enter the number of rows

5



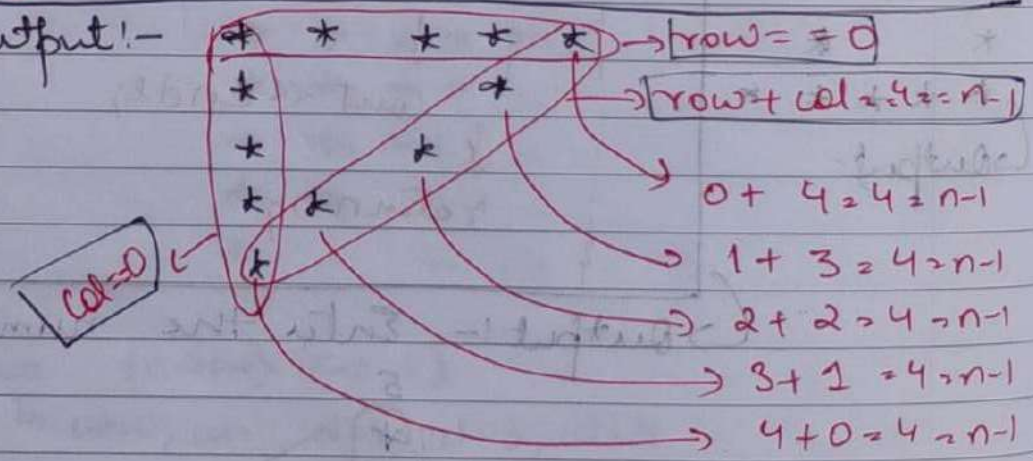

↳ In combining the 3 conditions we get the desired output.

## (8) Inverted Hollow Half Pyramid (H/W)

```
int main (){
    int n;
    cout<<"Enter the number of rows\n";
    cin>>n;
    for(int row=0; row<n; row++){
        for(int col=0; col<n-row; col++){
            if(col==0||row==0||row+col==n-1){
                cout<<" * ";
            }
            else{
                cout<<"   ";
            }
        }
        cout<<endl;
    }
    return 0;
}
```

```
* * * * *
*     *
*   *
* *
*
```

Output:-




Thus, on combining these 3 conditions, we get the desired output.

## (9) Numeric half pyramid.

```
int main(){
    int n;
    cout << "Enter the number of rows\n";
    cin >> n
    for (int row=0; row<n; row++){
        for (int col=0; col<row+1; col++){
            cout << col+1;
        }
        cout << endl;
    }
    return 0;
}
```

1
12
123
1234
12345

Output :- Enter the number of rows
5

| | | | | |
|---|---|---|---|---|
| 1 | | | | |
| 1 | 2 | | | |
| 1 | 2 | 3 | | |
| 1 | 2 | 3 | 4 | |
| 1 | 2 | 3 | 4 | 5 |
| ↑ | ↑ | ↑ | ↑ | ← col=4, 4+1=5 |
| col=0 | col=1 | col=2 | col=3 | |
| 0+1=1 | 1+1=2 | 2+1=3 | 3+1=4 | |

i.e for each column, we are printing col+1.

## (10) Numeric inverted pyramid

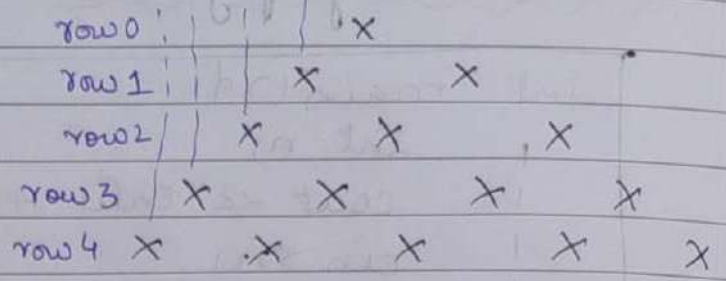
same logic but use the following logic to print inverted.

```
for (int row=0; row<n; row++){
    for (int col=0; col<n-row; col++){
        cout << col+1;
    }
}
```

# Live Class 6

## Let's Solve Some More Patterns.

### ① Full Pyramid




Thus, the outer loop will run 5 times i=0 to i=4
↳ 5 rows

Row 0:- 4 space 1 star
Row 1:- 3 space 2 star
Row 2:- 2 space 3 star
Row 3:- 1 space 4 star
Row 4:- 0 space 5 star

| Row | Space | formula | Star | formula |
|---|---|---|---|---|
| 0 | 4 | 5-0-1 | 1 | Row+1 |
| 1 | 3 | 5-1-1 | 2 | Row+1 |
| 2 | 2 | 5-2-1 | 3 | Row+1 |
| 3 | 1 | 5-3-1 | 4 | Row+1 |
| 4 | 0 | 5-4-1 | 5 | Row+1 |

↳ n-Row-1
n is the total number of rows

↳ Row+1

```
#include<iostream>
using namespace std;

int main(){
    int n;
    cout<<"Enter the number of rows\n";
```

```
cin >> n;
for (int i=0; i<n; i++){
    for (int j=0; j<n-i-1; j++){
        cout << " ";
    }
    for (int j=0; j<i+1; j++){
        cout << "* ";
    }
    cout << "\n";
}
return 0;
}
```

↳ Output:- Enter the number of rows:-
5

```
    *
   * *
  * * *
 * * * *
* * * * *
```

② Inverted Pyramid.

```
* * * * * → row 0
 * * * *  → row 1
  * * *   → row 2
   * *    → row 3
    *     → row 4
```

| Row | Space | Formula | Star | Formula |
|---|---|---|---|---|
| 0 | 0 | 0=0 | 5 | 5-0=5 |
| 1 | 1 | 1=1 | 4 | 5-1=4 |
| 2 | 2 | 1=1 | 3 | 5-2=3 |

↳ row = col ↳ col = n-row

```
#Include<iostream>
using namespace std;
int main() {
    int n;
    cout<<"Enter the value of n";
    cin>>n;
    for(int row=0; row<n; row++){
        for(int col=0; col<row; col++){
            cout<<" ";
        }
        for(int col=0; col<n-row; col++){
            cout<<"* ";
        }
        cout<<"\n";
    }
    return 0;
}
```

↳ Output:- Enter the value of n
5

```
* * * * *
 * * * *
  * * *
   * *
    *
```

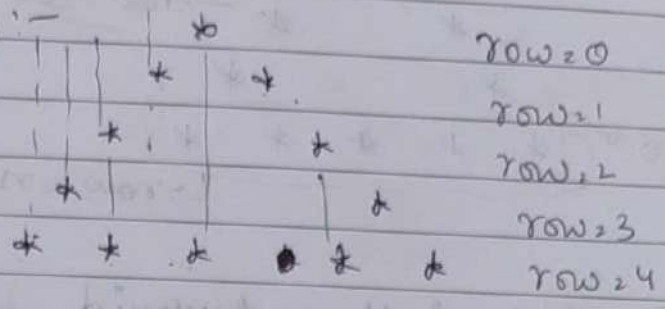
(3) Diamond Pattern (imp)

```
    *
   * *
  * * *
 * * * *
* * * * *
 * * * *
  * * *
   * *
    *
```

i.e basically diamond is a combination of full pyramid and full inverted pyramid.

We already coded full pyramid and inverted full pyramid.

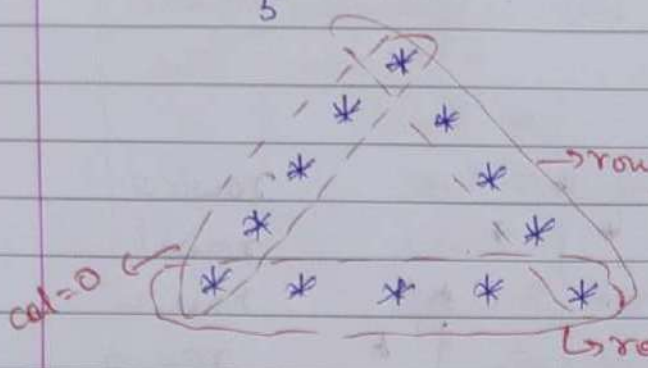
④ Hollow Pyramid :- (imp)




```
#include <iostream>
using namespace std;

int main() {
    int n;
    cout << "Enter the number of rows \n";
    cin >> n;
    for (int row=0; row<n; row++) {
        for (int col=0; col<n-row-1; col++) {
            cout << " ";
        }
        for (int col=0; col<row+1; col++) {
            if (col==0 || row==n-1 || col==row) {
                cout << "* ";
            }
            else {
                cout << "  ";
            }
        }
        cout << "\n";
    }
    return 0;
}
```

Output:- Enter the number of rows

5




(5) Inverted hollow pyramid.

row 0 → * * * * * 0 space 5 star
row 1 → * * 1 space 2 star
row 2 → * * 2 space 2 star
row 3 → * * 3 space 2 star
row 4 → * 4 space 1 star

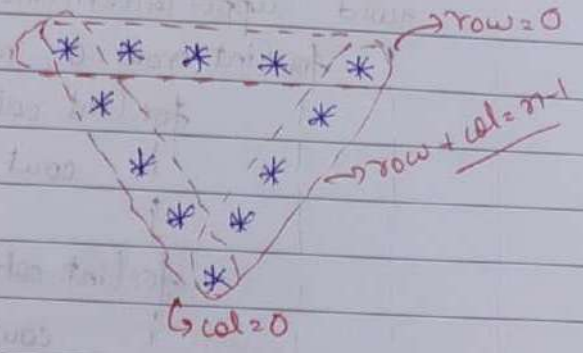
```
#include<iostream>
using namespace std;
int main(){
    int n;
    cout<<"Enter the number of rows\n";
    cin>>n;
    for(int row=0; row<n; row++){
        for(int col=0; col<row; col++){
            cout<<" ";
        }
        for(int col=0; col<n-row*; col++){
            if(row==0 ||row+col==n-1||col==0){
                cout<<"* ";
            }
            else{
                cout<<" ";
            }
        }
```

```
    cout<<"\n";
  }
  return 0;
}
```

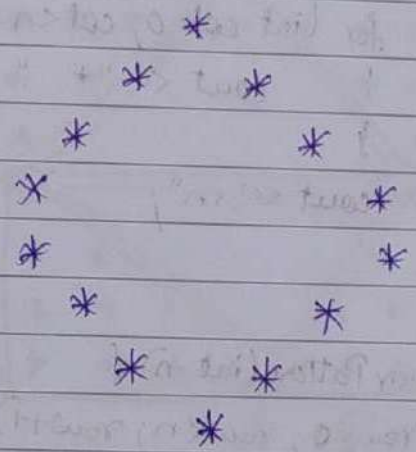
→ Output :-

Enter the number of rows

5




⑥ Hollow Diamond.



A hollow diamond is just a combination of hollow pyramid and an inverted hollow pyramid

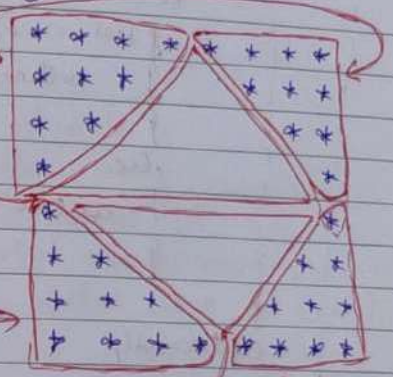
⑦ Mix Pyramid.

```
row0  * * * * * * * *   ← row 0 (1 space)
row1  * * *       * * * ← row 1 (4 space)
row2  * *           * * ← row 2 (8 space)
row3  *               * ← row 3 (12 space)
row0  *               * ← row 4 (12 space)
row1  * *           * * ← row 5 (8 space)
row2  * * *       * * * ← row 6 (4 space)
row3  * * * *   * * * * ← row 7 (1 space)
```

```
#include<iostream>
using namespace std;
void upperPattern(int n){
    for(int row=0; row<n; row++){
        for(int col=0; col<n-row; col++){ //inverted pyramid
            cout<<"* ";
        }
        for(int col=0; col<row*4; col++){ //spaceful pyramid
            cout<<" ";
        }
        for(int col=0; col<n-row; col++){ //inverted flipped pyramid
            cout<<"* ";
        }
        cout<<"\n";
    }
}
void lowerPattern(int n){
    for(int row=0; row<n; row++){
        for(int col=0; col<row+1; col++){ //pyramid
            cout<<"* ";
        }
        for(int col=0; col<(n-row)*4-4; col++){ //space
            cout<<" ";
        }
        for(int col=0; col<row+1; col++){ //pyramid
            cout<<"* ";
        }
        cout<<"\n";
    }
}
int main(){
    int n;
    cout<<"Enter the number of rows\n";
    cin>>n;
    upperPattern(n/2);
    lowerPattern(n/2);
    return 0;
}
```

↳ Output:- Enter the number of rows

8



⑧ Fancy 1-2 pattern

```
1
2 * 2
3 * 3 * 3
4 * 4 * 4 * 4
5 * 5 * 5 * 5 * 5
```

```
#include <iostream>
using namespace std;
int main(){
    int n;
    cout << "Enter the number of rows\n";
    cin >> n;
    for(int row=0; row<n; row++){
        for(int col=0; col<row+1; col++){
            cout << row+1;
            if (col==row){
                continue;
            }
            else
                cout << " * ";
        }
        cout << endl;
    }
}
```

Output :- Enter the number of rows
5
1
2 * 2
3 * 3 * 3
4 * 4 * 4 * 4
5 * 5 * 5 * 5 * 5

⑨ Full Fancy 1 2 pyramid

```
1
2 * 2
3 * 3 * 3
4 * 4 * 4 * 4
5 * 5 * 5 * 5 * 5
4 * 4 * 4 * 4
3 * 3 * 3
2 * 2
1
```

```
#include<iostream>
using namespace std;
void upperPattern(int n){
    for(int row=0; row<n; row++){
        for(int col=0; col<row+1; col++){
            cout<<row+1;
            if(col==row){
                continue;
            }
            else{
                cout<<" * ";
            }
        }
        cout<<endl;
    }
}
void lowerPattern(int n){
    for(int row=n-1; row>0; row--){
        for(int col=0; col<row; col++){
            cout<<row;
            if(col==row-1){
                continue;
            }
        }
```

```
        else
            cout << " * ";
        }
    }
    cout << endl;
  }
}
int main() {
    int n;
    cout << "Enter the number of rows\n";
    cin >> n;
    upperPattern(n/2);
    lowerPattern(n/2);
    return 0;
}
```

↳ Output:- Enter the number of rows

```
10
1
2 * 2
3 * 3 * 3
4 * 4 * 4 * 4
5 * 5 * 5 * 5 * 5
4 * 4 * 4 * 4
3 * 3 * 3
2 * 2
1
```

10 ABCBA Pattern

```
A
A B A
A B C B A
A B C D C B A
A B C D E D C B A
```

```
#include <iostream>
using namespace std;
int main() {
    int n;
    cout<<"Enter the number of rows\n";
    cin>>n;
    for(int row=0; row<n; row++) {
        char a=65;
        for(int col=0; col<row+1; col++) {
            cout<<a<<" ";
            a++;
        }
        a--;
        while(a>65) {
            a--;
            cout<<a<<" ";
        }
        cout<<endl;
    }
}
```

Output:- Enter the number of rows

```
5
A
A B A
A B C B A
A B C D C B A
A B C D E D C B A
```

# Live class 7



## Bitwise Operators & Functions

### Applications of bitwise operators

(1) Finding odd and even numbers

We know that if (n%2==0) then n is an even number but the modulo operator (%) is a very expensive operator in terms of resources and so we use the bitwise (&) and operator

> n & 1 = 0 when n is even
> n & 1 = 1 when n is odd

Eg:- (1) 4 & 1

```
4 = 0100
1 = 0001
4&1 = 0000
```

i.e 4 & 1 = 0

because 4 is even

(2) 5 & 1

```
5 = 0101
1 = 0001
    0001
```

i.e 5 & 1 = 1

because 5 is odd.

> Logic:- An odd number has LSB = 1 and hence, its & with 1 gives 1. Similarly,
> An even number has LSB=0 and hence, its & with 1 gives 0.

(2) Finding unique numbers.

We can use the bitwise XOR operation to find a unique number among many numbers

Eg 1:- 3^3

3 = 011
3 = 011
3^3 = 000

ie 3^3 = 0 so 3 is not unique

Eg 2:- 3^4

3 = 011
4 = 100
3^4 = 111

3^4 ≠ 0

so both 3 and 4 are unique

Eg ③ :- 4^5

4 = 100
5 = 101
4^5 001

∴ both 4 & 5 are unique

Eg:- Find the unique elements from the following

3, 8, 7, 4, 4, 12, 3, 12, 8

To find unique elements, we will do

3^8^7^4^4^12^3^12^8

Same elements cancel out after XOR operation, so

~~3~~^~~8~~^7^~~4~~^~~4~~^~~12~~^~~3~~^~~12~~^~~8~~

∴ 7 is a unique element

③ Multiply the number by power of 2

5 << 1

5 = 0000 0101 = 5 = 5
5<<1 = 0000 1010 = 10 = 5×2
5<<2 = 0001 0100 = 20 = 5×2^2 or 10×2
5<<3 = 0010 1000 = 40 = 5×2^3 or 20×2

Thus, on doing a left shift operation, the number gets multiplied by 2 each time

④ Dividing the number by power of 2

20>>1 ⇒ 20 = 0001 0100

20>>1 = 0000 1010 = 10 = $20/2 = 20/2^1$

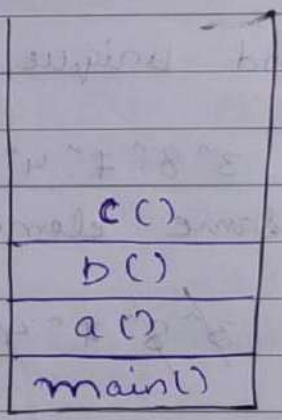
20>>2 = 0000 0101 = 5 = $20/4 = 20/2^2$

20>>3 = 0000 0010 = 2 = $20/8 = 20/2^3$

Thus, on doing a right shift operation, the number is divided each time by 2.

Function Call Stack

```
int main(){
        a();
    } return 0;
int a(){
        b();
        return 3;
    }
int b(){
        c();
        return 2;
int c(){
        return 1;
    }
```




Function call stack

↓

Last in First out

The execution of the ~~functions~~ code starts from the main() and hence the first entry in the stack is main().

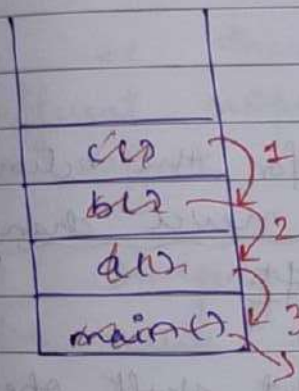
The function main has the function call to

the function a() and hence second entry is a();

The function a() has the function call b() and hence third entry is b();

The function b() calls the function c() and hence the fourth entry is c();

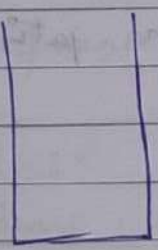
Stack is a last in first out datastructure and hence the last entry i.e c() will be the first to be taken out or popped.




Each function returns a value which we never used in the calling function.

The functions are removed from the call stack when all the lines of code of that particular function is executed or a return statement is executed.

Finally,



the function call stack is now empty which represents that no more functions are left to be executed and the entire code is successfully executed.

Very Very Imp

**Does right shifting a negative number give a positive number?**

No, right shifting a negative number will not give a positive number because

Consider an int → 4B = 32 bits

range = $-2^{31}$ to $2^{31}-1$ (signed int)

i.e we need 31 bits to represent this range

However, int is a 32 bit number

The MSB = 1 for negative and
MSB = 0 for positive

and the 31 bits are used for the actual digit. Thus, the sign bit is never changed while performing right shift.

Thus, on performing a right shift operation, a negative number will not become a positive number.

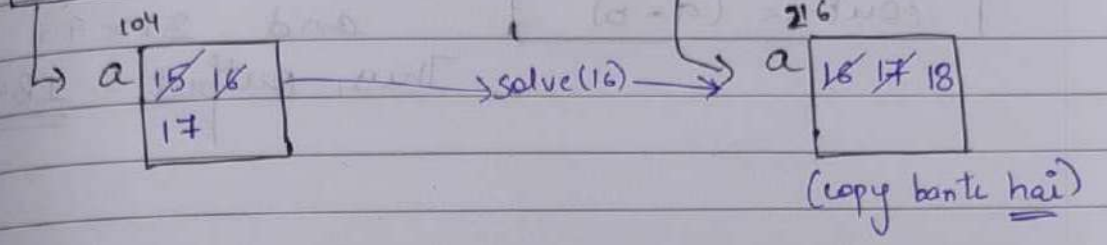
**Pass by value / Pass by reference**

(variables ki copy banti hai) (actual variables ki value change ho jati hai)

① Pass by Value

```
int main(){
    int a = 15;
    a++;
    solve(a); //pass by value
    a++;
    cout << a;
}
```

```
void solve(int a){
    a++;
    cout << a;
    a++;
}
```




(copy banti hai)

Output :- 17 17

i.e these two are independant variables in different scopes and changing one variable will not affect the other variable.

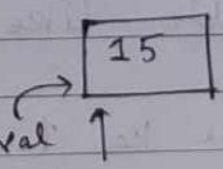
⇒ Changing the value of a in main() changes the value of the variable a in the main function

⇒ Changing the value of a in solve() changes the value of the variable a in the solve() function

② Pass by reference

int &love
↳ reference variable

```
int val = 15;
int &demo = val;
```

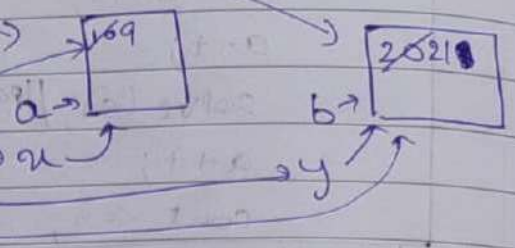



i.e this memory location can be accessed by using any of val or demo.

Eg ①

```
int a=10;
int b=20;
int &x=a;
int &y=b;
x--;
b++;
cout<<(a*b)
```

→ Output :- 189



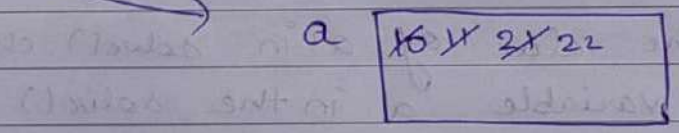

and 21×9 = 189

Thus, output = 189

Eg ②

```
int main(){
    int a=10;
    a++;
    solve(a);
    a++;
    cout<<a;
}
```

```
void solve(int &a){
    a=a+10;
    return
}
```




∴ Output = 22

i.e In pass by reference, all the modifications are performed on the same variable and not on two copies of the same variable like in pass by value.

Jab answer ka track rakhna ho tb use pass by reference.

M T W T F S S

Page No.:

Date:

YOUVA

# Basic Programming Notes

- Ravinder Jetarwal

# Data Structures

- Ravinder Jetarwal