Intro to Programming & Flowcharts

03 January 2022 15:26

Topics covered:

- → Course Flow
- → Flow charts
- → Pseudocode

Q. How to solve a programming problem?

→ Given the problem,

1. Understand the problem. → Add 2 numbers
2. Check the given values. → 2 variables. Data Types?
3. Figure out an approach → a+b = my answer.
   - This comes from practice and past coding experience.
4. Code! → `int ans = a+b; cout << ans << endl;`

WARNING ⚠

Agar code samajh na aaye to

Aage sab kuch cover hoga.

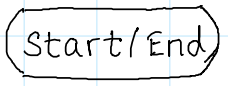
Given some problem [P], say you 'think' of some solution, ki aise aise karenge, etc. Now write down this crude solution on paper, not necessarily in correct syntax (code ki bhasha). Now your idea is on paper. Convert this rough work, also called 'pseudocode' into a program in a programming language of your choice, say C++.

Pseudocode : A very simple and high-level (upar-upar ka) form of computer language that is used in program design.

A Flowchart is a diagramatic representation of an approach. This draws out all the steps of your approach in order.

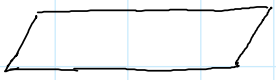
Components :

① Terminator : Specifies the start and end of a program.



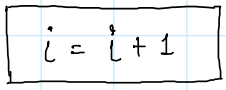
→ Terminator

② Parallelogram : For taking input or showing output.

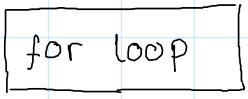


Input / Output

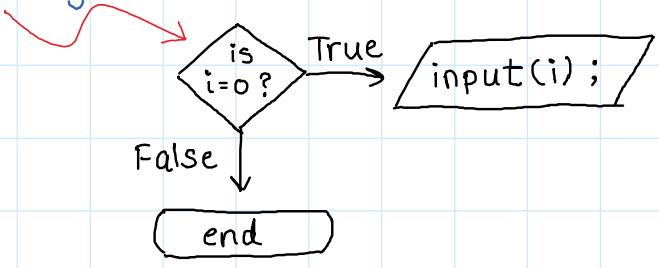
③ Process : Operations and processes ke liye.



or



④ Decision Making : (Diamond Shape)

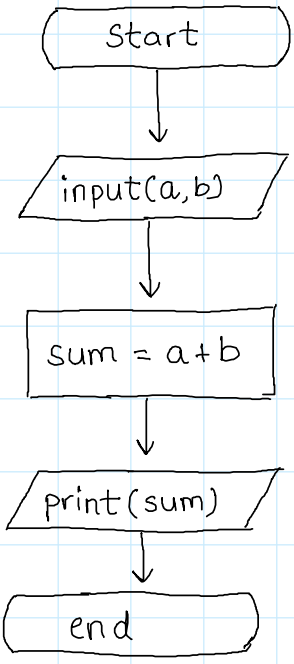


⑤ Circle : Connectors (To be covered when we discuss functions / methods)

⑥ Arrows : Code ka pravaah dikhane ke liye.
(Upar flowchart dekho ↑)

EXAMPLE :

Flow chart for adding 2 numbers.



Pseudocode for adding 2 numbers:

→ input 2 numbers a and b.
→ Let sum = a+b
→ Print out sum

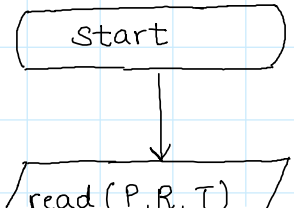
→ Read a and read b.
→ Sum variable is a+b
→ sum chaapo

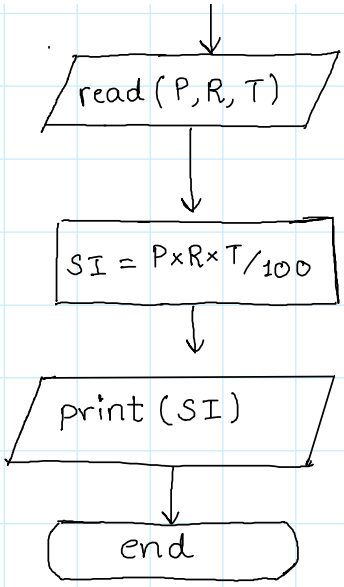
Both pseudocodes are OK. No pseudocode is wrong as long as the logic is same / similar.

EXAMPLE: Calculate simple interest.

$$SI = \frac{P \times R \times T}{100}$$

(P - Principal Amount
R - Rate of Interest
T - Time)





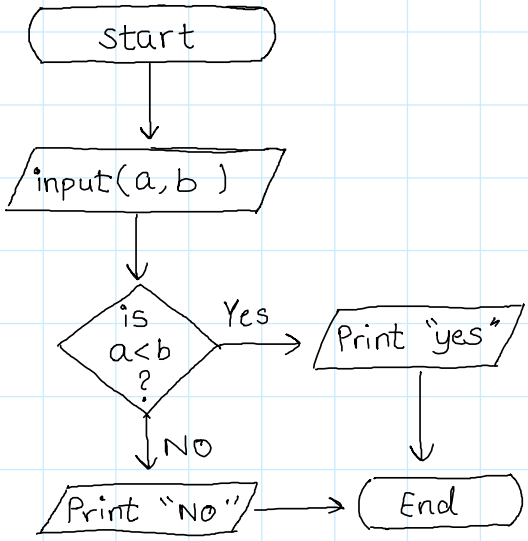
Pseudocode

→ read P, R and T

→ Make SI = P×R×T/100

→ Print SI

EXAMPLE: Determine if a < b



PSEUDOCODE:

→ Read a and b.

→ if a<b

then print Yes

else

print No

NEW CONCEPT : % (modulo) operator

Gives the remainder after division a/b.

∴ a % b = Remainder of a/b.

Eg : 5 % 2 = 1

6 % 4 = 2

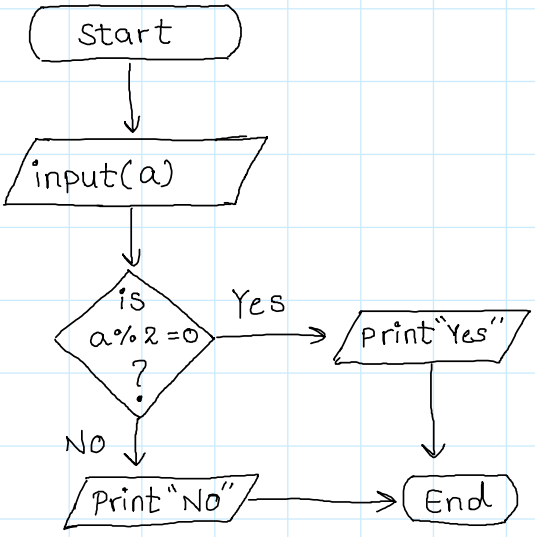
8 % 4 = 0

* 4 % 9 = 4 when a > b, a % b = a.

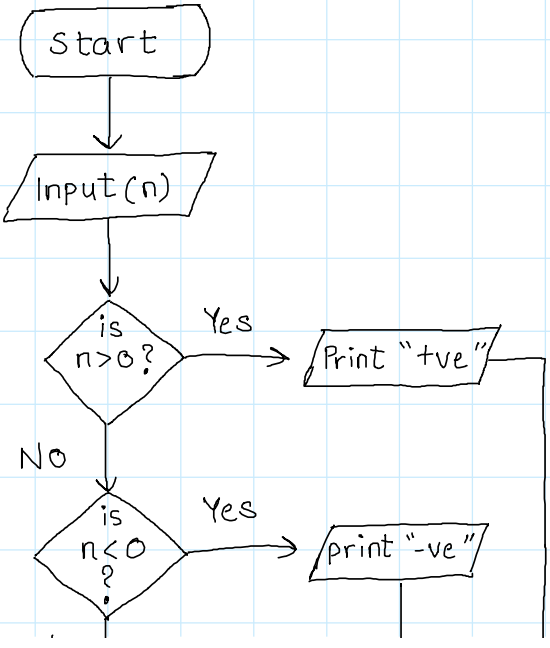
∴ If n % 2 = 0, n is even

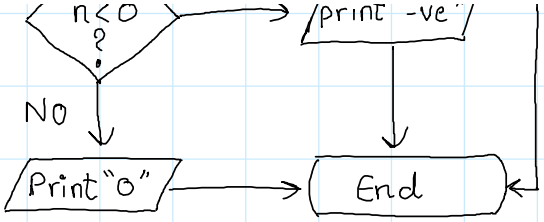
else n is odd.

EXAMPLE : Check if n is even or odd.



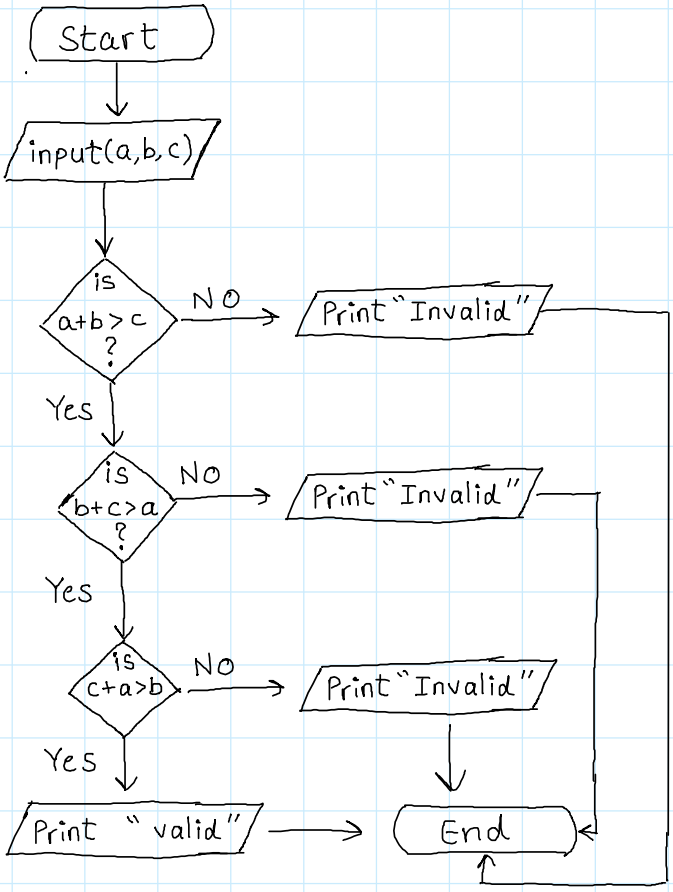
EXAMPLE : Is n positive, negative or zero.





Homework : Check if a given triangle is valid.

Hint - $a+b>c$, $b+c>a$ & $c+a>b$



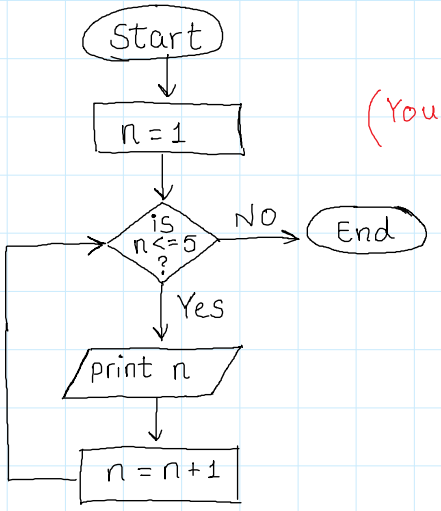
Loops

03 January 2022 17:39

Let Variable n = 1.

Now make n go from 1 to 5.

Soln.

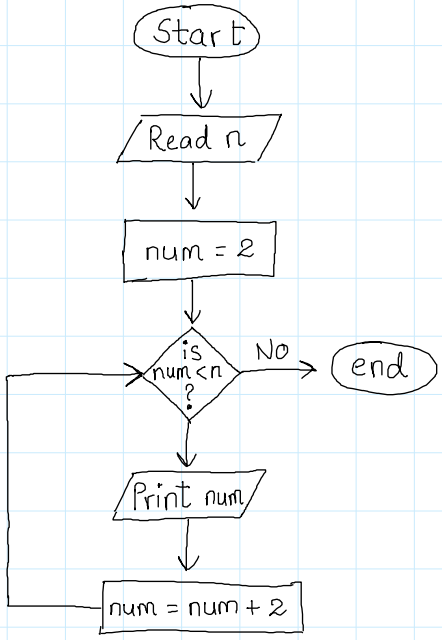


(You can also give input n = 1)

This is where we use loops to continuously perform some action while updating some value.

EXAMPLE: Print even numbers b/w 1 and n. (exclusive)

Hint - Even numbers start from 2 and occur alternatively.



Pseudocode

→ Input(n)

→ Let num = 2

→ while num < n,
print num
num = num+2

→ End

EXAMPLE: Print all odd numbers from 1 to n (inclusive)
(Homework hai)

Pseudocode

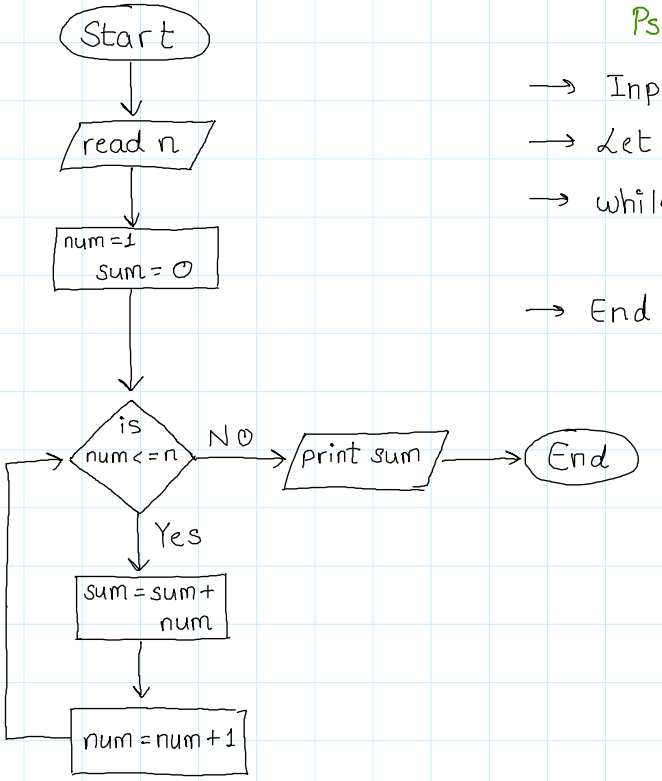
→ Input (n)

→ Let a = 1

→ while a < 1, print(a)
a = a + 2.

→ End

EXAMPLE: Find sum from 1 to n.



Pseudocode

→ Input (n)

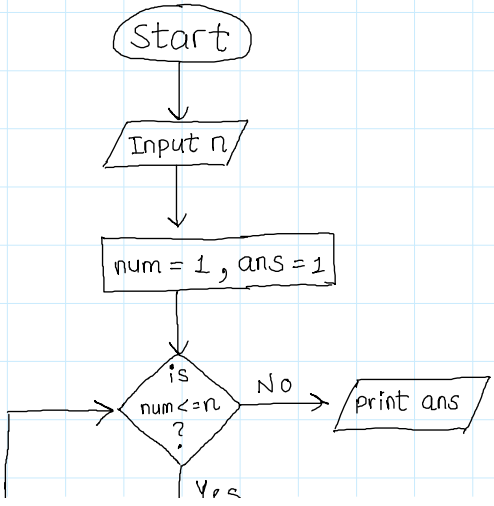
→ Let num = 1 and sum = 0

→ while num <= n,
sum = sum + num
num = num + 1

→ End

Homework: Find n!

Hint - $n! = n \times (n-1) \times (n-2) \times \ldots \times 3 \times 2 \times 1$



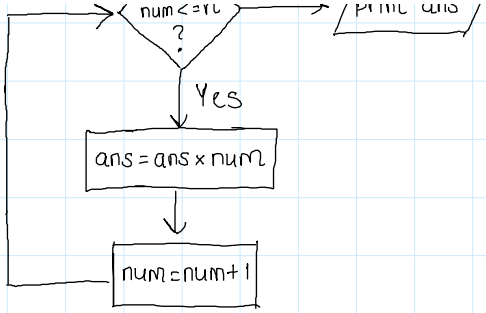
Pseudocode

→ Input (n)

→ Let num = 1 and ans = 1

→ while num <= n,
ans = ans × num
num = num + 1

→ End

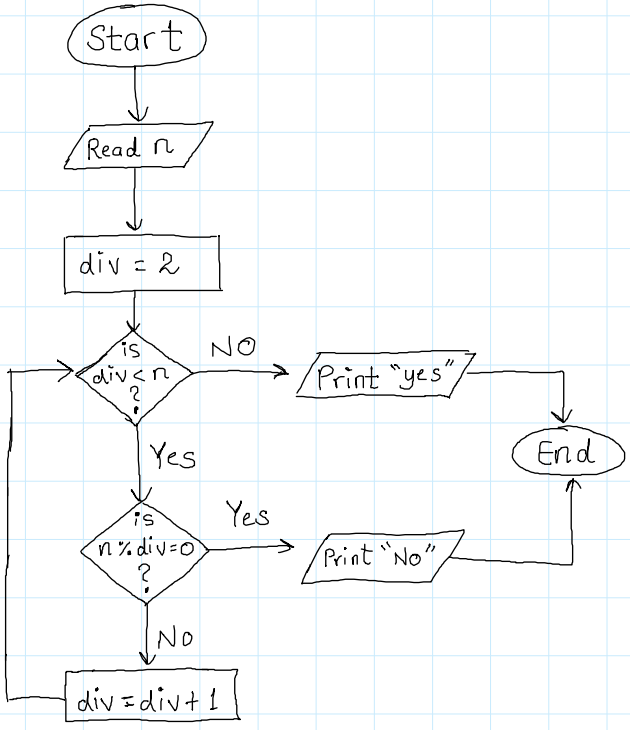


Example: Check if n is prime.

Input: 5 Output: Yes

Input: 9 Output: NO

Prime n : n % (any number from 2 to n-1) $\neq 0$ → Written as != in C++



Pseudocode

→ Input (n)

→ Let div = 2

→ while div < 2,

if n % div = 0

Print "No"

exit

else

div = div + 1

→ Print "Yes"

Programming Language

03 January 2022 18:33

What is a Programming Language ?

→ Jaise khaana khaane ke liye mummy/papa ko bolna padhta hai, vaise hi we must instruct our computer to perform some task for us.

→ More formally, a programming language is a way to communicate with a computer. It is a formal language which consists of sets of strings that produce various kinds of machine output.

Eg: C, C++, Java, Python, R, GO, etc.

```
#include<iostream>

using namespace std;

// main function -
// where the execution of program begins
int main()
{
    // prints hello world
    cout<<"Hello World";

    return 0;
}
```

Output:

```
Hello World
```

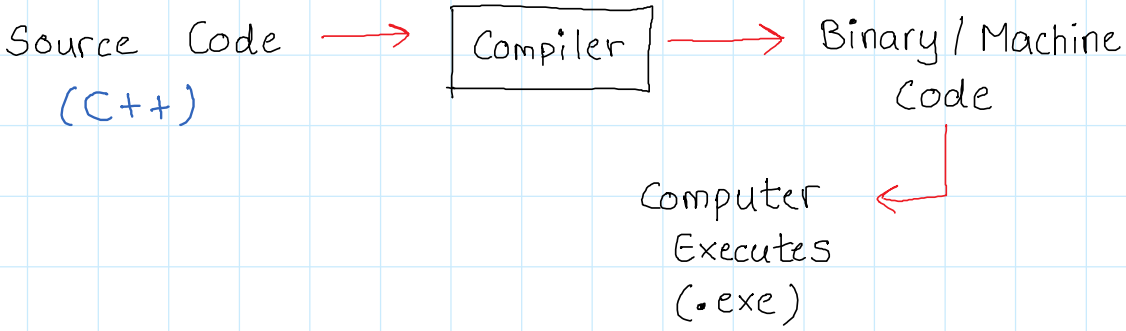
⇒ This code in C++ instructs our computer to print "Hello World" on our screen.

Source: https://www.geeksforgeeks.org/writing-first-c-program-hello-world-example/

→ Every language must be written following some rules called syntax of that language.

→ A computer essentially only understands binary codes of 0s and 1s. A compiler processes the statements of a programming language into Machine Code (Binary).

Source Code → [Compiler] → Binary / Machine

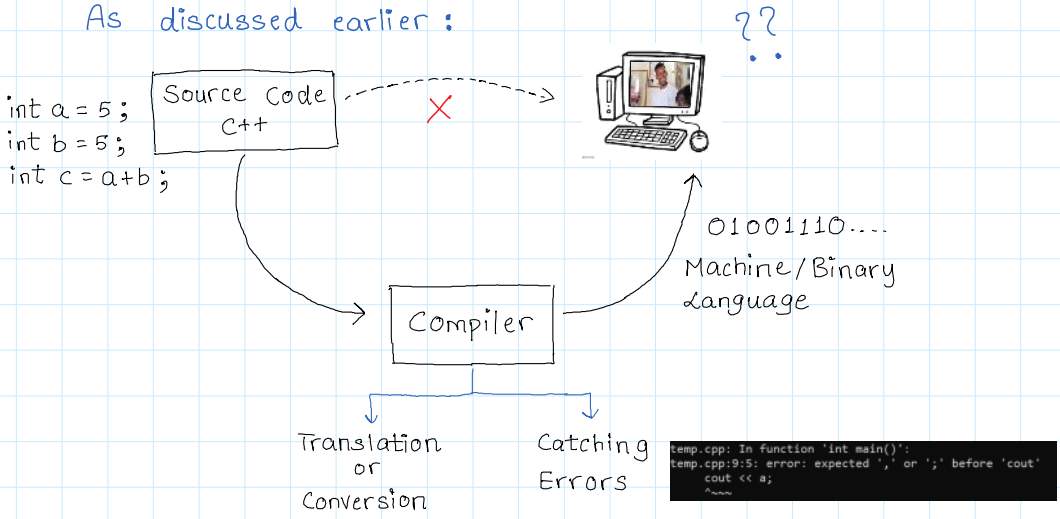


Read more about Compilers here: https://www.geeksforgeeks.org/introduction-to-compilers/

First Program in C++

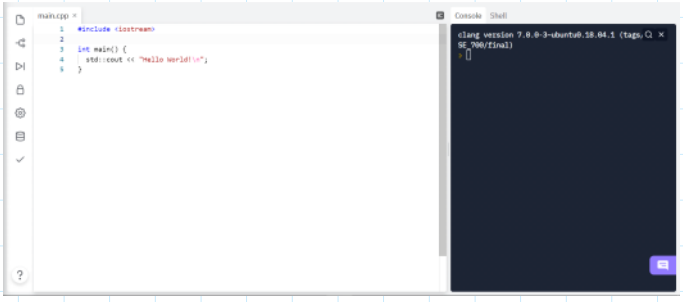
04 January 2022 20:53

As discussed earlier :



IDEs (Integrated Development Environment) :

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



First Program in C++ : (Namaste Duniya)

① Our program will find `int main(){ }` and start executing from there.

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

More about scope of a function

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

④ This cout is already defined in a file (header file)

which must be included before using cout.

⇒ `#include <file_name>` is a preprocessor directive which runs before the program is compiled and includes the file to be used later in the source code.

A file called iostream has cout defined in it so:

`#include <iostream>`

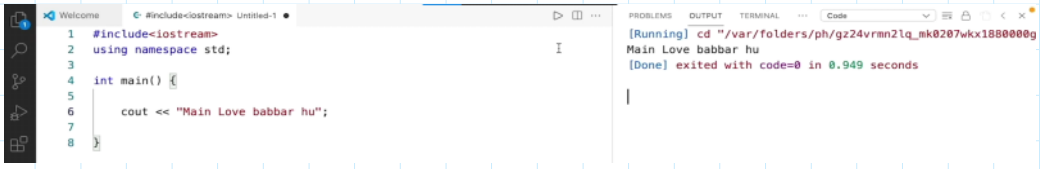
Hint: i/o means input/output.

⑤ Namespaces : Stack Overflow Question

`using namespace std;`

⑥ Using cout :

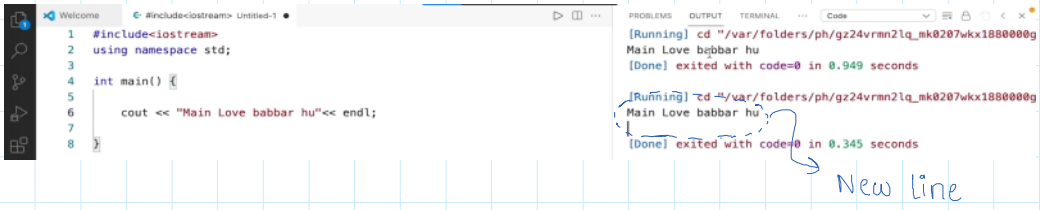
We use `<<` after `cout` to display something to Standard Output (your screen) within std namespace.



⑦ endl : Used to enter new line. Just like ENTER.

endl is like "\n" which is a new line escape sequence character used in various languages including C++.

cout << " Namaste Duniya " << endl ;



⑧ endl; ; is used to terminate statements.

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

Eg- int: Stores integers like -5, 0, 8, etc.
char: Single character like 'a', '+', '$', '7', etc.
float: Floating point values like -2.014, 1.0000, 6.7800

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

Data Type	Meaning	Size (in Bytes)
int	Integer	2 or 4
float	Floating-point	4
double	Double Floating-point	8
char	Character	1
wchar_t	Wide Character	2
bool	Boolean	1
void	Empty	0

Source: Programiz

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

```
char ch = 'a';
```

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

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

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

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

Know more about C++ Data Types

Variable Naming / Nomenclature :

1. Can contain alphabets, numbers and underscores.
2. Cannot start with a number.
3. Cannot be keywords like int, cout, double, bool, etc.
4. Case sensitive
5. Cannot contain special symbols like %, $, !, #, etc.

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

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

```
ning] cd "/var/folders/ph/gz24vrmn2lq_mk0207wkx1880000gn/T/"
CodeRunnerFile.cpp:10:10: error: redefinition of 'a' with a
char a = 'v';
         ^
CodeRunnerFile.cpp:6:9: note: previous definition is here
int a = 123;
    ^
```

Using different data types in code :

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

int main() {

    int a = 123;

    cout<< a << endl;

    char b = 'v';

    cout<< b <<endl;

    bool bl = true;
    cout<< bl << endl;

    float f = 1.2;
    cout<< f << endl;

    double d = 1.23;
    cout<< d << endl;

}
```

```
[Running] cd "/var/folders/ph/gz24vrmn2lq_mk0207wkx1880000g
123
v
1

[Done] exited with code=0 in 0.595 seconds

[Running] cd "/var/folders/ph/gz24vrmn2lq_mk0207wkx1880000g
123
v
1
1.2

[Done] exited with code=0 in 0.551 seconds

[Running] cd "/var/folders/ph/gz24vrmn2lq_mk0207wkx1880000g
123
v
1
1.2
1.23

[Done] exited with code=0 in 0.537 seconds
```

Check the size of different Data Types for your system using sizeof(variable_name);

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

int main() {
    int a = 4;
    double b = 1.90;
    char c = '+';
    int sizeInteger = sizeof(a), sizeDouble = sizeof(b), sizeChar = sizeof(c);
    cout << "Size of an integer is " << sizeInteger << " bytes" << endl;
    cout << "Size of a double is " << sizeDouble  << " bytes" << endl;
    cout << "Size of a char is " << sizeChar  << " bytes" << endl;
}
```

```
clang++-7 -pthread -std=c++17 -o main main.cpp
./main
Size of an integer is 4 bytes
Size of a double is 8 bytes
Size of a char is 1 bytes
```

HOW IS DATA STORED IN MEMORY ?

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

In binary, 8 = 1000 (4 bits needed)

∴ int a → 00000000 00000000 00000000 00001000 } 32 bits

Eg: int b = 5;

b

5

4 bytes.

address = 100 (assume)

100, 101, 102, 103

4 bytes are consumed

Eg: char c = 'a';

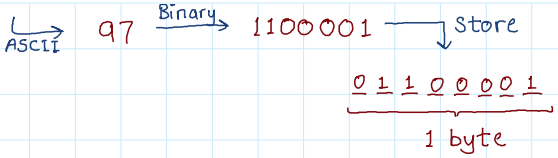
Characters are mapped to the standard ASCII values

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

ASCII Table

(Homework)

char c = 'a';


ASCII
97
Binary
1100001
Store
0 1 1 0 0 0 1
1 byte

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

Example :

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

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

Example :

```
char ch = 98;
```

98 will automatically get typecasted to its corresponding character.

This automatic typecasting is called implicit typecasting.

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

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

→

```
[Running] cd "/var/folders/ph/gz24vrmn2lq_mk0207wkx1880000g
97
b
```

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

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

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

→

```
[Running] cd "/var/folders/ph/gz24vrmn2lq_mk0207wkx1880000g
tempCodeRunnerFile.cpp:33:16: warning: implicit conversion
    char ch1 = 123456;
           ~~   ^~~~~~
1 warning generated.
```

How are negative numbers stored ?

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

First Bit → 0 means Positive
→ 1 means Negative

Steps to store -5 in binary format :

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

Example : a = -5.

1. -5 → 5 (Ignore the -ve sign)
2. 5 → 0101 → 00.....0101 (Binary)
   (29 Zeros)
3. 2's complement = 1's Complement + 1.

```
        5  →  00.....0101
1's compl. →  11.....1010   (Flip the bits)
            +             1
2's Compl. →  11.....1011
```

2's Compl. → 11.....1011 (29 Ones) + 1

Displaying Negative Number :

① Take 2's complement of the stored number

Stored : 11.....1011

This shows -ve

11....1011

1's comp. 00....0100

2's comp. 00...0101 = 5

-5 print ho gaya !

Why 2's Complement ?

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

1	0	0	-----	0	0

and

0	0	0	-----	0	0

will be equal & thus waste space.

Store only Positive Integers

The default signed representation allows us to store both positive & negative values.
To store only positive integers, we use unsigned..

Eg: unsigned int a = 10;

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

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

Output :
4294967184 ??

Explaination :
We tried to store -112.

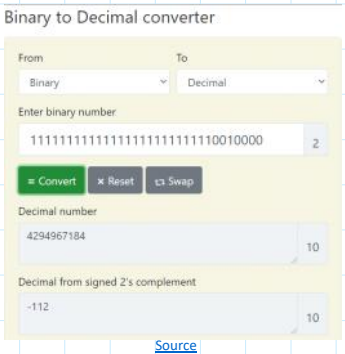
-112 = 2's Complement of 112.

112 = 00.....01110000.
25 Zeros.

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

2's Compl. = 11.....10010000

Unsigned int uses all 32 bits to store the value and the MSB (=1) will make the value.
An unsigned int does not use the 2's complement again to display the number.
Thus, 11....10010000. gets printed as it is in decimal.
25 ones



Source

Therefore,

```
unsigned int a = -112;

cout<< a << endl;
```

→

```
[Running] cd "/var/folders/ph/gz24vrmn2lq_mk0207wkx1880000g
4294967184
```

Operators

05 January 2022 19:05

Basic Arithmetic Operators:

+ , - , * , / , %

Caution ⚠

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

Examples: 5/2 = 1.

3/5 = 0

9/2 = 4

② int / float } float
float / int }

double / int } double
int / double }

cout << 5.0/2 << endl;

Output:

2.5

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

```
[Running] cd "/var/folders/ph/gz24vrmn2lq_mk0207wkx1880000g
0.4
0.4

[Done] exited with code=0 in 0.591 seconds
```

Relational Operators:

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

① Is a = b ?

a==b → Yes → 1
No → 0

② Is a greater than or equal to b ?

a >= b → Yes → 1
No → 0

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

```
[Running] cd "/var/folders/ph/gz24vrmn2lq_mk0207wkx1880000g
0

[Done] exited with code=0 in 0.851 seconds

[Running] cd "/var/folders/ph/gz24vrmn2lq_mk0207wkx1880000g
0
0
1
0
1
1

[Done] exited with code=0 in 1.124 seconds
```

Logical Operators:

&& , || , !

(AND) (OR) (NOT)

① Logical AND:

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

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

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

Output :

1

② Logical OR :

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

Example :
```
int a=5, b=10, c=15;
cout <<((a>5) || (b<10) || (c>=15));
```

Output :

1

③ Logical NOT :

Inverts the logic. True ⇌ False

Non-zero ⇌ Zero

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

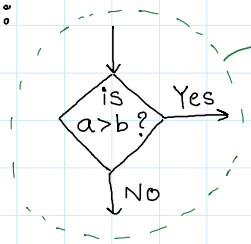
Output :

0

1

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

Recall :



→ In code is called a conditional statement.

Example : I/P : a, b

O/P : $\begin{cases} a & \text{if } a > b \\ b & \text{otherwise} \end{cases}$

Conditionals like these are solved using if statements.

```
if (~~~~) {
    ~~~~
    ~~~~
    ~~~~
}
```

If the condition within ( ) is true, then execute the entire code block within { }.

Thus,

```
if ( a > b ) {
    cout << a << endl;
}
if ( a <= b ) {
    cout << b << endl;
}
```

In the above example, instead of checking again for b > a in the second if block, we know that if the first if condition does not get fulfilled, then the second block must be executed no matter what. This can be achieved using an if - else block.

If this is fulfilled, else block won't execute

```
if ( a > b ) {
    cout << a << endl;
}
else {
    cout << b << endl;
}
```

If if block doesn't execute, then this else block will execute.

NEW CONCEPT :

`cin >> n;` → Waits for user to give input and assigns it to n at its address.

Example- `int a;`
`cin >> a;`

① variable a will get initialized with a random, 'garbage' value.

② Program waits for user to give input (an integer) to a and stores a with that value.

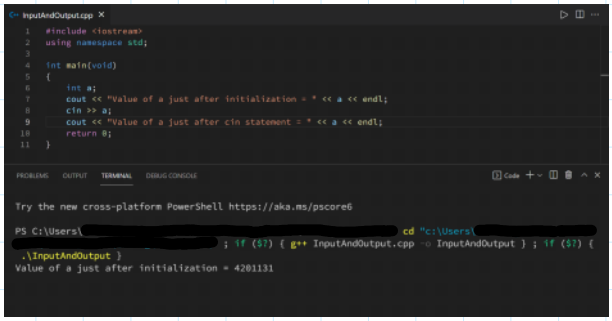


Fig ①

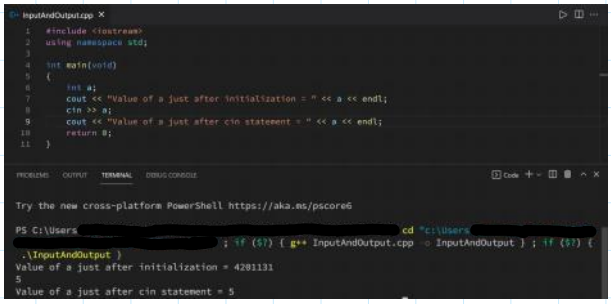


Fig ②

Back to if-else :

Here, the if block doesn't execute and the program exits without printing anything

①

InputAndOutput.cpp

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

int main(void)
{
    int a;
    cin >> a;

    if(a > 0) {
        cout << "a is positive" << endl;
    }
    return 0;
}
```

```
PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE
{ .\InputAndOutput }
5
a is positive
```

②

InputAndOutput.cpp

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

int main(void)
{
    int a;
    cin >> a;

    if(a > 0) {
        cout << "a is positive" << endl;
    }
    return 0;
}
```

```
PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE
-1
PS [redacted]
```

③

InputAndOutput.cpp

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

int main(void)
{
    int a;
    cin >> a;
```

④

InputAndOutput.cpp

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

int main(void)
{
    int a;
    cin >> a;
```

```cpp
int main(void)
{
    int a;
    cin >> a;

    if(a > 0) {
        cout << "a is positive" << endl;
    }
    else {
        cout << "a is not positive" << endl;
    }
    return 0;
}
```

```
PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE
 .\InputAndOutput }
5
a is positive
```

```cpp
int main(void)
{
    int a;
    cin >> a;

    if(a > 0) {
        cout << "a is positive" << endl;
    }
    else {
        cout << "a is not positive" << endl;
    }
    return 0;
}
```

```
PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE
 .\InputAndOutput }
-2
a is not positive
```

Program to compare two numbers :

InputAndOutput.cpp

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

int main()
{
    int a, b;
    cout << "Enter the value of a: ";
    cin >> a;
    cout << "Enter the value of b: ";
    cin >> b;

    if(a > b) {
        cout << "a is greater than b" << endl;
    }
    if (b > a) {
        cout << "b is greater than a" << endl;
    }
    return 0;
}
```

```
PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE
 .\InputAndOutput }
Enter the value of a: 6
Enter the value of b: 4
a is greater than b
```

InputAndOutput.cpp

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

int main()
{
    int a, b;
    cout << "Enter the value of a: ";
    cin >> a;
    cout << "Enter the value of b: ";
    cin >> b;

    if(a > b) {
        cout << "a is greater than b" << endl;
    }
    if (b > a) {
        cout << "b is greater than a" << endl;
    }
    return 0;
}
```

```
PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE
 .\InputAndOutput }
Enter the value of a: 3
Enter the value of b: 6
b is greater than a
```

Note : cin ignores ENTER (\n), TAB (\t) and SPACE ( ) while taking input. These are called whitespace characters. Use cin.get() to read these whitespace characters.

Using nested if - else :

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

int main()
{
    int a;
    cin >> a;

    if(a > 0) {
        cout << "a is positive" << endl;
    }
    else {
        cout << "a is not positive" << endl;
    }
    return 0;
}
```

→ In this else block, we can see that either a will be negative or 0. So we can further break down this else block.

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

int main()
{
    int a;
    cin >> a;
    if(a > 0) {
        cout << "a is positive" << endl;
    }
    else {
        if (a == 0) {
            cout << "a is 0" << endl;
        }
        else {
            cout << "a is negative" << endl;
        }
    }
    return 0;
}
```

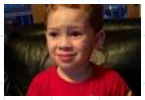
```
PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE
 .\InputAndOutput }
5
a is positive
```

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

int main()
{
    int a;
    cin >> a;
    if(a > 0) {
        cout << "a is positive" << endl;
    }
    else {
        if (a == 0) {
            cout << "a is 0" << endl;
        }
        else {
            cout << "a is negative" << endl;
        }
    }
    return 0;
}
```

```
PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE
 .\InputAndOutput }
0
a is 0
```

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

int main()
{
    int a;
    cin >> a;
    if(a > 0) {
        cout << "a is positive" << endl;
    }
    else {
        if (a == 0) {
            cout << "a is 0" << endl;
        }
        else {
            cout << "a is negative" << endl;
        }
    }
    return 0;
}
```

```
PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE
 .\InputAndOutput }
-9
a is negative
```



Code ganda hote jaa raha hai. Ek-do baar aur nested if-else kar diya to coding chorni padhegi.

Solution : else if

```
if ( ~~~~ ) {

}
else if ( ~~~~ ) {

}
else {

}
```

```cpp
int a ;
cout<<" enter the value of a "<<endl;
cin>>a;


if(a>0) {
    cout<<" A is positive"<< endl;
}
else if(a<0) {
    cout<<" A is negative"<<endl;
}
else {
    cout<<" A is 0"<<endl;
}
```

Note : else if and else are optional.
else can be used as a default case.

Homework : Output ??

(1)

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

int main() {

        int a = 9;
        if(a == 9) {
                cout<< "NINEY";
        }

        if(a > 0){
                cout<<"POSITIVE";
        }
        else
        {
                cout<<"NEGATIVE";
        }
}
```

Answer : NINEYPOSITIVE

(2)

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

int main() {

        int a = 2;
        int b = a+1;

        if((a=3)==b) {
                cout<<a;
        }
        else
        {
                cout<<a+1;
        }
}
```

Answer : 3

a is assigned 3.
Now, a == b is true.

(3)

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

int main() {

        int a = 24;

        if(a > 20){
                cout<< "Love ";
        }
        else if(a == 24) {
                cout<<"Lovely";
        }
        else
        {
                cout<<"Babbar";
        }
        cout<<a;
}
```

Answer : Love24

(4) Code Homework :

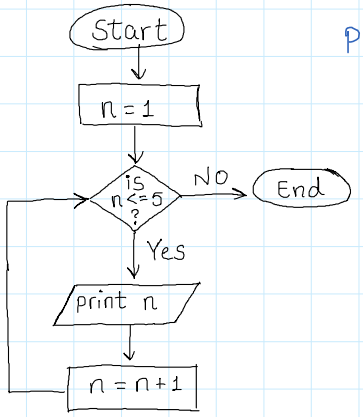
```cpp
#include <iostream>
using namespace std;

int main()
{
    char a;
    cin >> a;
    // 'A' is 65
    // 'a' is 97
    // '0' is 48
    if(a >= 'A' && a <= 'Z') {
        cout << "This is upper case" << endl;
    }
    else if(a >= 'a' && a <= 'z') {
        cout << "This is lower case" << endl;
    }
    else if(a >= '0' && a <= '9') {
        cout << "This is a digit" << endl;
    }
    return 0;
}
```

We can also use ASCII values

Loops

07 January 2022 20:16

Recollect from flowcharts :



Print from 1 to 5.

while loop :

```
while (condition) {
}
```

जब तक ये तुरू हैं

तब तक ये करते रहो

while the condition is true, keep on executing the block

Example :

Print 1 to N.

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

int main()
{
    int n;
    cin >> n;
    int i = 1;
    while(i <= n) {
        cout << i << " ";
        i = i + 1;
    }
    return 0;
}
```

```
PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE
.\InputAndOutput }
8
1 2 3 4 5 6 7 8
```

Example : Find sum from 1 to n.

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

int main()
{
    int n;
    cin >> n;
    int sum = 0;
    int i = 1;
    while(i <= n) {
        sum = sum + i;
        i = i + 1;
    }
    cout << "Sum from " << 1 << " to " << n << " = " << sum << endl;
    return 0;
}
```

```
PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE
 { .\InputAndOutput }
8
Sum from 1 to 8 = 36
```

Example: Find sum of all even numbers from 1 to n.

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

int main()
{
    int n;
    cin >> n;
    int sum = 0;
    int i = 2;
    while(i <= n) {
        if(i % 2 == 0)
            sum = sum + i;
        i = i + 1;
    }
    cout << "Sum from " << 1 << " to " << n << " = " << sum << endl;
    return 0;
}
```

```
PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE
8
Sum from 1 to 8 = 20
```

Homework: Fahrenheit to Celsius.

$$C = \frac{5}{9}(F - 32)$$

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

int main()
{
    float fahrenheit;
    cin >> fahrenheit;
    float celsius = (5.0/9) * (fahrenheit - 32);
    cout << fahrenheit << " F = " << celsius << " C" << endl;

    return 0;
}
```

```
PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE
 { .\InputAndOutput }
100
100 F = 37.7778 C
```

Example: Prime or not ?

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

int main()
{
    int n;
    cin >> n;
    int i = 2;
    while(i < n) {
        if(n % i == 0) {
            cout << "Not Prime" << endl;
            return 0;
        }
        i = i + 1;
    }
    cout << "Prime" << endl;
    return 0;
}
```

```
PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE
 { .\InputAndOutput }
14
Not Prime
```

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

int main()
{
    int n;
    cin >> n;
    int i = 2;
    while(i < n) {
        if(n % i == 0) {
            cout << "Not Prime" << endl;
            return 0;
        }
        i = i + 1;
    }
    cout << "Prime" << endl;
    return 0;
}
```

```
PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE
 { .\InputAndOutput }
11
Prime
```

Patterns

07 January 2022 20:46

Example :

n = 4 (4 rows, 4 columns)

For every row, print 'row' num. of columns or stars.

here n = 3

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

int main() {
    int n;
    cin >> n;
    int i = 0;
    while(i < n) { // i = 0, 1, 2, 3, ..., n-1  which is basically n times
        int j = 0;
        while(j < n) {
            cout << "* ";
            j = j + 1;
        }
        cout << endl; // after printing one row, we need to enter the next row so endl
        i = i + 1;
    }
    return 0;
}
```

```
PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE

 .\InputAndOutput }
3
* * *
* * *
* * *
```

Ye code samajh nahi aa raha

Don't worry, initially thoda difficult lag sakta hai BUT the next video has 18 problems solved end to end so keep going

Example : For n = 3,

```
1 1 1
2 2 2
3 3 3
```

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

int main() {
    int n;
    cin >> n;
    int i = 1;
    while(i <= n) {
        int j = 1;
        while(j <= n) {
            cout << i << " ";
            j = j + 1;
        }
        cout << endl;
        i = i + 1;
    }
    return 0;
}
```

```
PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE

4
1 1 1 1
2 2 2 2
3 3 3 3
4 4 4 4
```

Pattern Questions (Part- 2)

09 January 2022 19:06

Q1. Print

```
1 2 3 4
1 2 3 4
1 2 3 4
1 2 3 4
```

for n = 4

Answer :

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

int main() {
    int n;
    cin >> n;
    int i = 1; // Try printing in reverse (solved in the video)
    while(i <= n) {
        int j = 1;
        while(j <= n) {
            cout << j << " ";
            j = j + 1;
        }
        cout << endl;
        i = i + 1;
    }
    return 0;
}
```

```
PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

4
1 2 3 4
1 2 3 4
1 2 3 4
1 2 3 4
```

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

int main() {
    int n;
    cin >> n;
    int i = 1;
    // Try printing in reverse (solved in the video)
    while(i <= n) {
        int j = 1;
        while(j <= n) {
            cout << j << " ";
            j = j + 1;
        }
        cout << endl;
        i = i + 1;
    }
    return 0;
}
```

```
PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

rk\Coding\Patterns\" ; if ($?) { g++ Pattern.cpp -o Pattern
3
1 2 3
1 2 3
1 2 3
```

Homework : For n = 4,

```
4 3 2 1
4 3 2 1
4 3 2 1
4 3 2 1
```

Q2. Print :

```
1 2 3
4 5 6
7 8 9
```

for n = 3

Logic - maintain a variable, incrementing it by 1 after every cout.

Answer :

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

int main() {
    int n;
    cin >> n;
    int i = 1;
    int toPrint = 1;
    while(i <= n) {
        int j = 1;
        while(j <= n) {
            cout << toPrint << " ";
            toPrint = toPrint + 1;
            j = j + 1;
        }
        cout << endl;
        i = i + 1;
    }
    return 0;
}
```

```
PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

\Work\Coding\Patterns\" ; if ($?) { g++ P
3
1 2 3
4 5 6
7 8 9
```

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

int main() {
    int n;
    cin >> n;
    int i = 1;
    int toPrint = 1;
    while(i <= n) {
        int j = 1;
        while(j <= n) {
            cout << toPrint << " ";
            toPrint = toPrint + 1;
            j = j + 1;
        }
        cout << endl;
        i = i + 1;
    }
    return 0;
}
```

```
PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

rk\Coding\Patterns\" ; if ($?) { g++ Pa
4
1 2 3 4
5 6 7 8
9 10 11 12
13 14 15 16
```

Try printing :

```
9 8 7
6 5 4
3 2 1
```

for n = 3.

3 2 1

16 15 14 13
12 11 10 9
8 7 6 5
4 3 2 1

{Hint : Starting point has a relation with n

Q3. Print :

for n = 4.

{Hint : We are printing '*', row number of times where row = 1, 2, ...

Answer :

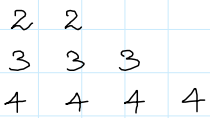
```cpp
#include <iostream>
using namespace std;

int main() {
    int n;
    cin >> n;
    int row = 1;
    while(row <= n) {
        int column = 1;
        while(column <= row) {
            cout << "* ";
            column = column + 1;
        }
        cout << endl;
        row = row + 1;
    }
    return 0;
}
```

```
PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE
rk\Coding\Patterns\" ; if ($?) { g++ Patter
4
*
* *
* * *
* * * *
```

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

int main() {
    int n;
    cin >> n;
    int row = 1;
    while(row <= n) {
        int column = 1;
        while(column <= row) {
            cout << "* ";
            column = column + 1;
        }
        cout << endl;
        row = row + 1;
    }
    return 0;
}
```

```
PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE
5
*
* *
* * *
* * * *
* * * * *
```

Q4. Print :



- for n - 4.

(Easy)

Answer :

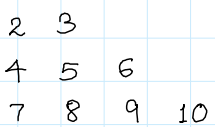
```cpp
#include <iostream>
using namespace std;

int main() {
    int n;
    cin >> n;
    int row = 1;
    while(row <= n) {
        int column = 1;
        while(column <= row) {
            cout << row << " ";
            column = column + 1;
        }
        cout << endl;
        row = row + 1;
    }
    return 0;
}
```

```
PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE
rk\Coding\Patterns\" ; if ($?) { g++ Pattern.cpp
4
1
2 2
3 3 3
4 4 4 4
```

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

int main() {
    int n;
    cin >> n;
    int row = 1;
    while(row <= n) {
        int column = 1;
        while(column <= row) {
            cout << row << " ";
            column = column + 1;
        }
        cout << endl;
        row = row + 1;
    }
    return 0;
}
```

```
PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE
5
1
2 2
3 3 3
4 4 4 4
5 5 5 5 5
```

Homework - Print :



for n = 4.

Answer :

```cpp
#include <iostream>
```

7 8 9 10

Answer :

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

int main() {
    int n;
    cin >> n;
    int row = 1, toPrint = 1;
    while(row <= n) {
        int column = 1;
        while(column <= row) {
            cout << toPrint << " ";
            toPrint = toPrint + 1;
            column = column + 1;
        }
        cout << endl;
        row = row + 1;
    }
    return 0;
}
```

```
PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE

4
1
2 3
4 5 6
7 8 9 10
```

q5 Print

```
1
2 3
3 4 5
4 5 6 7
```

for n = 4.

Logic - We are starting from i for every row i.
Keep incrementing i times for each row.

Answer :

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

int main() {
    int n;
    cin >> n;
    int row = 1, toPrint = 1;
    while(row <= n) {
        int column = 1;
        toPrint = row; // start printing from row
        while(column <= row) {
            cout << toPrint << " ";
            toPrint = toPrint + 1;
            column = column + 1;
        }
        cout << endl;
        row = row + 1;
    }
    return 0;
}
```

```
PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE

4
1
2 3
3 4 5
4 5 6 7
```

Homework : Solve the above question without using the extra variable toPrint.

Logic - Column starts from 1 for every row. Change it to start from row and go till row + row = row * 2 (exclusive)

Answer :

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

int main() {
    int n;
    cin >> n;
    int row = 1;
    while(row <= n) {
        int column = row;
        while(column < row * 2) {
            cout << column << " ";
            column = column + 1;
        }
        cout << endl;
        row = row + 1;
    }
    return 0;
}
```

```
PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE
rk\Coding\Patterns\" ; if ($?) { g++ Pattern.cp
4
1
2 3
3 4 5
4 5 6 7
```

Q6. Print

```
1
2 1
3 2 1
4 3 2 1
```

Hint: starting from row number and decrementing.

Logic :

	column = 1	2	3	4
row = 1	1			
row = 2	2	1		
row = 3	3	2	1	
row = 4	4	3	2	1

Subtract column number from row number and add 1.

(row - column + 1)

Answer :

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

int main() {
    int n;
    cin >> n;
    int row = 1;
    while(row <= n) {
        int column = 1;
        while(column <= row) {
            cout << (row - column + 1) << " ";
            column = column + 1;
        }
        cout << endl;
        row = row + 1;
    }
    return 0;
}
```

```
PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE
rk\Coding\Patterns\" ; if ($?) { g++ Pattern.cpp -o Pat
5
1
2 1
3 2 1
4 3 2 1
5 4 3 2 1
```

q7 Print : A A A for n = 3.
B B B
C C C

Answer :

```cpp
int main() {
    int n;
    cin >> n;
    int i = 1;
    while(i <= n) {
        int j = 1;
        while(j <= n) {
            // you can also print 'A' + i - 1 because with an int
            // 'A' will get typecasted to 65
            cout << (char)(65 + i - 1) << " ";
            j = j + 1;
        }
        i = i + 1;
        cout << endl;
    }
    return 0;
}
```

```
PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE
rk\Coding\Patterns\" ; if ($?) { g++ Pattern.cpp -o Pattern } ; if ($?) {
4
A A A A
B B B B
C C C C
D D D D
```

q8. Print : A B C for n = 3
A B C
A B C (Homework Code)

Answer :

```cpp
int main() {
    int n;
    cin >> n;
    int i = 1;
    while(i <= n) {
        int j = 1;
        char ch = 'A';
        while(j <= n) {
            cout << ch << " ";
            ch = ch + 1;
            j = j + 1;
        }
        i = i + 1;
        cout << endl;
    }
    return 0;
}
```

```
PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE
rk\Coding\Patterns\" ; if ($?) { g++ Pattern.cpp -o
3
A B C
A B C
A B C
```

q9 Print : A B C for n = 3
D E F
G H I

Logic is same as with numbers

Answer :

```cpp
int main() {
    int n;
    cin >> n;
    int i = 1;
    char ch = 'A';
    while(i <= n) {
        int j = 1;
        while(j <= n) {
            cout << ch << " ";
            ch = ch + 1;
            j = j + 1;
        }
        i = i + 1;
        cout << endl;
    }
    return 0;
}
```

```
PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE
rk\Coding\Patterns\" ; if ($?) { g++ Pattern.cpp -o Pa
3
A B C
D E F
G H I
```

Q10 Print :

A B C
B C D
C D E

Logic is same as with numbers

{ 'A' + row - 1
for row = 1, 2, ...
increment this

Answer :

```cpp
int main() {
    int n;
    cin >> n;
    int i = 1;
    while(i <= n) {
        char ch = 'A' + i - 1;
        int j = 1;
        while(j <= n) {
            cout << ch << " ";
            ch = ch + 1;
            j = j + 1;
        }
        i = i + 1;
        cout << endl;
    }
    return 0;
}
```

```
PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE
4
A B C D
B C D E
C D E F
D E F G
```

```cpp
int main() {
    int n;
    cin >> n;
    int i = 1;
    while(i <= n) {
        int j = 1;
        while(j <= n) {
            char ch = 'A' + i + j - 2;
            cout << ch << " ";
            j = j + 1;
        }
        i = i + 1;
        cout << endl;
    }
    return 0;
}
```

```
PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE
rk\Coding\Patterns\" ; if ($?) { g++ Pattern.cpp -o Patterr
4
A B C D
B C D E
C D E F
D E F G
```

Q11. Print :

A
B B
C C C

for n = 3

Answer :

```
int main() {
    int n;
    cin >> n;
    int i = 1;
    while(i <= n) {
        int j = 1;
        char ch = 'A' + i - 1;
        while(j <= i) {
            cout << ch << " ";
            j = j + 1;
        }
        i = i + 1;
        cout << endl;
    }
    return 0;
}
```

```
PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE
rk\Coding\Patterns\" ; if ($?) { g++ Pattern.cpp -o
4
A
B B
C C C
D D D D
```

Q12 Print :

```
A
B C
D E F
G H I J
```

(Try yourself)

Logic same as with numbers homework question.

Answer :

```
int main() {
    int n;
    cin >> n;
    int i = 1;
    char ch = 'A';
    while(i <= n) {
        int j = 1;
        while(j <= i) {
            cout << ch << " ";
            ch = ch + 1;
            j = j + 1;
        }
        i = i + 1;
        cout << endl;
    }
    return 0;
}
```

```
PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE
4
A
B C
D E F
G H I J
```

Q13 Print :

```
A
B C
C D E
D E F G
```

for n = 4

Logic is same as numbers.

Answer:

```
int main() {
    int n;
    cin >> n;
    int i = 1;
    while(i <= n) {
        char ch = 'A' + i - 1;
        int j = 1;
        while(j <= i) {
            cout << ch << " ";
            ch = ch + 1;
            j = j + 1;
        }
        i = i + 1;
        cout << endl;
    }
    return 0;
}
```

```
PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE
rk\Coding\Patterns\" ; if ($?) { g++ Pattern.cpp
4
```

The Babbar Answer

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

int main() {

    int n;
    cin>>n;

    int row = 1;

    while(row <= n) {

        int col = 1;

        while(col <= row) {
            char ch = ('A' + row + col - 2);
            cout<<ch;
            col = col + 1;
        }
        cout<<endl;
        row = row + 1;

    }
```

```
20  }
```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

```
rk\Coding\Patterns\" ; if ($?) { g++ Pattern.cpp
4
A
B C
C D E
D E F G
```

```
18          col = col + 1;
19      }
20      cout<<endl;
21      row = row + 1;
22
23  }
24
25  }
```

(Homework)

Q 14 Print :

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

Logic - We are starting from ch = ('A' + rows - 1) in row 1.
ch = ('A' + rows - 2) in row 2.
ch = ('A' + rows - 3) in row 3.
⋮
ch = ('A' + rows - i) for row i.

Answer :

```
4   int main() {
5       int n;
6       cin >> n;
7       int i = 1;
8       while(i <= n) {
9           char ch = 'A' + n - i;
10          int j = 1;
11          while(j <= i) {
12              cout << ch << " ";
13              ch = ch + 1;
14              j = j + 1;
15          }
16          i = i + 1;
17          cout << endl;
18      }
19      return 0;
20  }
```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

```
rk\Coding\Patterns\" ; if ($?) { g++ Pattern.cpp
4
D
C D
B C D
A B C D
```

Q 15 Print :

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

for n = 4 (Notice the number of spaces and its relation with the row number, i)

Logic - for row i, we are first printing n-i spaces and then printing i stars.

Answer :

```
int main() {
    int n;
    cin >> n;
    int i = 1;
    while(i <= n) {
        int space = 1;
        while(space <= n - i) {
            cout << " ";
            space = space + 1;
        }
        int j = 1;
        while(j <= i) {
            cout << "*";
            j = j + 1;
        }
        i = i + 1;
        cout << endl;
    }
    return 0;
}
```

```
PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE
\Work\Coding\Patterns\" ; if ($?) { g++
4
   *
  **
 ***
****
```

Q16. Print :

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

(Homework)

Answer :

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

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

```
PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE
\Work\Coding\Patterns\" ; if ($?) { g++ Patte
4
* * * *
* * *
* *
*
```

Q17. Print :

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

for n = 4

(Homework)

Logic : Printing i-1 spaces & n-i+1 stars for i^{th} row.

Answer :

```
int main() {
    int n;
    cin >> n;
    int i = 1;
    while(i <= n) {
        int j = 1, space = 1;
        while(space < i) {
            cout << " ";
            space = space + 1; // space++ is also used
        }
        while(j <= n - i + 1) {
            cout << "*";
            j = j + 1;
        }
        i = i + 1;
        cout << endl;
    }
    return 0;
}
```

```
PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE
\Work\Coding\Patterns\" ; if ($?) { g++ Pattern.cpp -o Pattern }
4
****
 ***
  **
   *
```

Q18 Print :

```
1 1 1 1
  2 2 2
    3 3
      4
```

for n = 4 (Homework)

Answer :

```
4  int main() {
5      int n;
6      cin >> n;
7      int i = 1;
8      while(i <= n) {
9          int j = 1, space = 1;
10         while(space < i) {
11             cout << " ";
12             space = space + 1; // space++ is also used
13         }
14         while(j <= n - i + 1) {
15             cout << i;
16             j = j + 1;
17         }
18         i = i + 1;
19         cout << endl;
20     }
21     return 0;
22 }
```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

```
\Work\Coding\Patterns\" ; if ($?) { g++ Pattern.cpp -o Pat
4
1111
 222
  33
   4
```

Q19 Print :

```
      1
    2 2
  3 3 3
4 4 4 4
```

for n = 4 (Homework)

Answer :

```
4  int main() {
5      int n;
6      cin >> n;
7      int i = 1;
8      while(i <= n) {
9          int j = 1, space = 1;
10         while(space <= n - i) {
11             cout << " ";
12             space = space + 1; // space++ is also used
13         }
14         while(j <= i) {
15             cout << i;
16             j = j + 1;
17         }
18         i = i + 1;
19         cout << endl;
20     }
21     return 0;
22 }
```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

```
\Work\Coding\Patterns\" ; if ($?) { g++ Pattern.cpp -o Pat
4
   1
  22
 333
4444
```

Q20 Print :

```
      1
    2 3
  4 5 6
7 8 9 10
```

for n = 4

Logic - Print space (n-i) times and print num. Then increment num.

Answer -

```cpp
int main() {
    int n;
    cin >> n;
    int i = 1, num = 1;
    while(i <= n) {
        int j = 1, space = 1;
        while(space <= n - i) {
            cout << "  "; // Notice two spaces
            space = space + 1;
        }
        while(j <= i) {
            cout << num << " ";
            num = num + 1;
            j = j + 1;
        }
        i = i + 1;
        cout << endl;
    }
    return 0;
}
```

```
PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE
4
      1
    2 3
  4 5 6
7 8 9 10
```

Q 21 Print :

```
      1
    1 2 1
  1 2 3 2 1
1 2 3 4 3 2 1
```

for n = 4

Logic - Use 2 while loops for each row i.
One for printing (n-i) "__" (double spaces)
One for printing from 1 to i.
Other for printing from (i-1) to 1.

Answer :

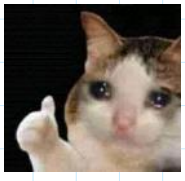
```cpp
#include <iostream>
using namespace std;

int main() {
    int n;
    cin >> n;
    int i = 1;
    while(i <= n) {
        int j = 1, space = 1;
        while(space <= n - i) {
            cout << "  "; // Notice two spaces
            space = space + 1;
        }
        while(j <= i) {
            cout << j << " ";
            j = j + 1;
        }
        j = i - 1;
        while(j >= 1) {
            cout << j << " ";
            j = j - 1;
        }
        i = i + 1;
        cout << endl;
    }
    return 0;
}
```

TAGDA HOMEWORK :

Print :

```
1 2 3 4 5 5 4 3 2 1
1 2 3 4 ☆ ☆ 4 3 2 1
1 2 3 ☆ ☆ ☆ ☆ 3 2 1
1 2 ☆ ☆ ☆ ☆ ☆ ☆ 2 1
1 ☆ ☆ ☆ ☆ ☆ ☆ ☆ ☆ 1
```

for n = 5



Answer :

```cpp
int main() {
    int n;
    cin >> n;
    int i = 1;
    while(i <= n) {
        // part 1: numbers from 1 to n - i + 1
        int j = 1;
        while(j <= n - i + 1) {
            cout << j << " ";
            j = j + 1;
        }
        // part 2: Stars (i-1)*2 times
        j = 1;
        while(j <= (i-1)*2) {
            cout << "* ";
            j = j + 1;
        }
        // part 3: numbers from n - i + 1 to 1
        j = n - i + 1;
        while(j >= 1) {
            cout << j << " ";
            j = j - 1;
        }
        i = i + 1;
        cout << endl;
    }
    return 0;
}
```

Bitwise Operators

11 January 2022 17:39

① Bitwise AND : (&)

Eg: 2 & 3 = 010 & 011 = 010

Both bits must be 1 to get 1.

x	y	AND
0	0	0
0	1	0
1	0	0
1	1	1

Eg: 5 & 7 :

```
  101
& 111
-----
  101   = (5)
```

② Bitwise OR : (|)

Eg: 2 | 5 :

```
  010
  101
-----
  111
```

If at least one of the bits is 1 then the output will be 1.

x	y	OR
0	0	0
0	1	1
1	0	1
1	1	1

Eg: 3 | 6 :

```
  011
  110
-----
  111
```

③ Bitwise NOT : (~)

Eg: (consider 4 byte representation)

a = 2 = 000...0010 (30 bits)

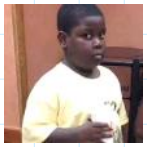
~a = ~2 = 111...1101

The leading 1 is the signed bit.

Two's complement :

```
  000...0010
+          1
------------
```

x	NOT
0	1
1	0



000....0011 = -3

④ Bitwise XOR : (^)

x	y	XOR
0	0	0
0	1	1
1	0	1
1	1	0

If different bits → 1
If same bits → 0

Eg: 5 ^ 7 = 101
^ 111
010 = 2

```
1  #include<iostream>
2  using namespace std;
3
4
5  int main() {
6
7      int a = 4;
8      int b = 6;
9
10     cout<<" a&b " << (a&b) << endl;
11     cout<<" a|b " << (a|b) << endl;
12     cout<<" ~a " << ~a << endl;
13     cout<<" a^b " << (a^b) << endl;
14
15 }
```

```
BitwiseOperators && "/Users/lovebabbar/"BitwiseOperators
 a&b 4
 a|b 6
 ~a -5
 a^b 2
lovebabbar@192 ~ %
```

⑤ Left Shift : (<<)

Eg: 5 << 1 → Left shift 5, 1 time
= 101 << 1 → 00....0101 << 1
= 00....1010 = 10 = 5×2^1

Eg: 3 << 2 → 00....011 << 2
= 00....1100 = 12 = 3×2^2

In most cases we multiply with power of 2.
But in some cases, this isn't true.

Eg: 0100...00 << 1 = 1000...00
Positive → Negative ??

So << is ok for smaller numbers.

⑥ Right Shift : (>>)

Eg: 15 >> 1 = 000....1111 >> 1
= 000....0111 = 7

Eg: 5 >> 2 = 000....0101 >> 2
000....0001 = (1)

Padding in << and >> for POSITIVE numbers is done with 0.

For NEGATIVE numbers, padding is compiler dependent.

```
cout<< (17>>1)<<endl;
cout<< (17>>2) <<endl;
cout<< (19<<1) <<endl;
cout<< (21<<2) <<endl;
```

```
8
4
38
84
lovebabbar@192 ~ %
```

(7) Increment / Decrement :

→ We can write $i = i+1$ as $i++$ or $++i$.

$i++$ is called Post - Increment
$++i$ is called Pre - Increment.

→ We can write $i = i-1$ as $i--$ or $--i$.

$i--$ is called Post - decrement
$--i$ is called Pre - decrement.

→ $a = a+b$ is same as $a += b$.

→ $a = a-b$ is same as $a -= b$.

Post - Increment : The value gets used first and then increments.

Eg: int i = 3, a = 2;

```
int sum = a + (i++);
sum = 2 + 3;
sum = 5;
```
Now i is 4.

Pre - Increment : The value gets incremented first and then gets used.

Eg: int i = 3, a = 2;

```
int sum = a + (++i);     i has become 4
sum = 2 + 4 ;            first.
sum = 6;
```

Post - Decrement : The value gets used first and then rements.

Eg: int i = 3, a = 2;

```
int sum = a + (i--);
sum = 2 + 3 ;
sum = 5;
  Now i is 2.
```

Pre - Decrement : The value gets remented first and then gets used.

Eg: int i = 3, a = 2;

```
int sum = a + (--i);     i has become 2
sum = 2 + 2 ;            first.
sum = 4 ;
```

```
20    int i = 7;
21
22    cout<< (++i) <<endl;
23    // 8
24    cout<< (i++) <<endl;
25    // 8 , i = 9
26    cout<< (i--) <<endl;
27    //9 , i = 8
28    cout<< (--i) <<endl;
29    // 7, i =7
```

```
8
8
9
7
lovebabbar@192 ~ %
```

Q1

```
1  #include <iostream>
2  using namespace std;
3  int main()
4  {
5          int a, b = 1;
6          a = 10;
7          if (++a)
8             cout << b;
9          else
10             cout << ++b;
11 }
```

Answer : 1

Q2

```
#include <iostream>
using namespace std;
int main()
{
    int a = 1;
    int b = 2;

    if(a-- > 0 && ++b > 2 ){
    cout << "Stage1 - Inside If ";
    } else{
    cout << "Stage2 - Inside else ";
    }
    cout << a << " " << b << endl;
}
```

Answer : Stage1 - Inside If 0 3

Q3

```
#include <iostream>
using namespace std;
int main()
{
    int a = 1;
    int b = 2;

    if(a-- > 0 || ++b > 2 ){
    cout << "Stage1 - Inside If ";
    } else{
    cout << "Stage2 - Inside else ";
    }
    cout << a << " " << b << endl;
}
```

Answer : Stage1 - Inside If 0 2

Hint: Only one of the conditions must be true for || so it won't check ++b > 2.

Q3

```
#include <iostream>
using namespace std;
int main()
{
        int number = 3;
        cout << (25 * (++number) );
}
```

Answer : 100

Q4

```
#include <iostream>
using namespace std;
int main()
{
    int a = 1;
    int b = a++;
    int c = ++a;
    cout << b ;
    cout << c;
}
```

Answer : 13

Example Code :

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

int main() {

    int n ;
    cout<<" enter the value of n" <<endl;
    cin >> n;

    cout<<"printing count from 1 to n" << endl;

    for(int i = 1; i<=n; i++) {
        cout<< i << endl;
    }


}
```

```
& "/Users/lovebabbar/"forLoop
 enter the value of n
5
printing count from 1 to n
1
2
3
4
5
lovebabbar@192 ~ %
```

Structure : `for ( int i = 0 ; i < n ; i++ )`

- `int i = 0` — initialization
- `i < n` — cond. to check before every loop/iteration
- `i++` — updation

```
{
  ~~~~
  ~~~~
}
```

Not Necessary to specify.

Example : for(;;;) = ?

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

int main() {

    int n ;
    cout<<" enter the value of n" <<endl;
    cin >> n;

    cout<<"printing count from 1 to n" << endl;
    int i = 1;
    for(; i<=n; ) {
        cout<< i << endl;
        i++;
    }


}
```

→

```
lovebabbar@192 ~ % cd "/Users/lovebabbar/" && g++ forLoop.cpp -o forLoop &
& "/Users/lovebabbar/"forLoop
 enter the value of n
5
printing count from 1 to n
1
2
3
4
5
lovebabbar@192 ~ %
```

You can declare all the 3 parts outside the paranthesis ().

WRONG :

```cpp
int i = 1;
for(;;; ) {
    if(i<=n) {
        cout<< i << endl;
    }

    i++;
}
```

→ 

The for loop doesn't know when to stop.

Solution : break ;

↳ Gets you out of the current loop.

```cpp
int n ;
cout<<" enter the value of n" <<endl;
cin >> n;

cout<<"printing count from 1 to n" << endl;
int i = 1;
for(; ; ) {
    if(i<=n) {
        cout<< i << endl;
    }
    else{
        break;
    }

    i++;
}
```

```
 enter the value of n
5
printing count from 1 to n
1
2
3
4
5
lovebabbar@192 ~ %
```

Example: Using multiple inits., conditions and updations.

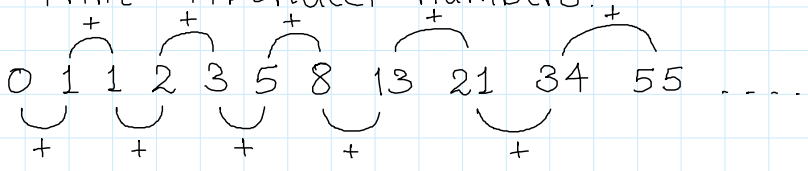
```cpp
for(int a = 0 , b =1, c = 2; a>=0 && b>=1 && c>=2; a--,b--, c-- ){
    cout<<a <<" "<< b << " "<<c << endl;
}
```

Question: Print 1 to n.

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

int main() {

    int n ;
    cout<<" enter the value of n" <<endl;
    cin >> n;

    int sum = 0;

    for(int i=1; i<=n; i++ ) {
        //sum = sum + i;
        sum += i;
    }

    cout<< sum <<endl;

}
```

```
lovebabbar@192 ~ % cd "/Users/lovebabbar/" && g++ forSum.cpp -o forSum &&
"/Users/lovebabbar/"forSum
 enter the value of n
5
15
lovebabbar@192 ~ %
```

Question: Print Fibonacci numbers.



Solution :

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

int main() {

    int n = 10;

    int a = 0;
    int b = 1;
    cout<<a <<" " <<b<<" ";
    for(int i = 1; i<=n; i++ ) {

        int nextNumber = a+b;
        cout<<nextNumber<<" ";

        a = b;
        b = nextNumber;
    }


}
```

```
lovebabbar@192 ~ % cd "/Users/lovebabbar/" && g++ forFib.cpp -o forFib &&
"/Users/lovebabbar/"forFib
0 1 1 2 3 5 8 13 21 34 55 89
lovebabbar@192 ~ %
```

Question: Print if prime. (Logic already covered)

Ex: I/P - 7
O/P - Yes

Solution:

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

int main() {

    int n ;
    cout<<" enter the value of n" <<endl;
    cin >> n;

    bool isPrime = 1 ;

    for(int i = 2; i<n; i++) {

        //rem = 0, Not a Prime
        if(n%i == 0) {
            //cout<<" Not a Prime Number" << endl;
            isPrime = 0;
            break;
        }
    }

    if(isPrime == 0) {
        cout<<" Not a Prime Number"<<endl;
    }
    else
    {
        cout<<"is a Prime Number "<<endl;
    }


}
```

```
lovebabbar@192 ~ % cd "/Users/lovebabbar/" && g++ forPrime.cpp -o forPrime
 && "/Users/lovebabbar/"forPrime
 enter the value of n
101
is a Prime Number
```

break ka brother continue

continue: Used to skip an iteration of the current loop.

It skips the remaining block of code for that iteration.

code example:

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

int main() {

    for(int i=0; i<5; i++) {

        cout<< " HI " << endl;
        cout<< " Hey " << endl;
        continue;

        cout<< "Reply toh karde " <<endl;

    }


}
```

```
lovebabbar@192 ~ % cd "/Users/lovebabbar/" && g++ continue.cpp -o continue
 && "/Users/lovebabbar/"continue
 HI
 Hey
 HI
 Hey
 HI
 Hey
 HI
 Hey
 HI
 Hey
lovebabbar@192 ~ %
```

Output Questions:

Q1.

```cpp
#include<iostream>
using namespace std;
int main() {

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

Output: 0 2 4

Q1.

```cpp
#include<iostream>
using namespace std;
int main() {

    for(int i = 0; i<=5; i++) {
        cout<< i << " ";
        i++;
    }

}
```

Output : 0 2 4

Q2.

```cpp
#include<iostream>
using namespace std;
int main() {

    for(int i = 0; i<=5; i--) {
        cout<< i << " ";
        i++;
    }

}
```

Output : 0 0 0 0 0 ..... (infinite times)

Q3.

```cpp
#include<iostream>
using namespace std;
int main() {

    for(int i = 0; i<=15; i += 2 ) {
        cout<< i << " ";

        if( i&1 ){
            continue;
        }

        i++;
    }
}
```

Output : 0 3 5 7 9 11 13 15

Q4

```cpp
#include<iostream>
using namespace std;
int main() {

    for(int i=0;i<5;i++) {
        for(int j=i;j<=5;j++) {
            cout<<i << " " << j <<endl;
        }
    }
}
```

Output :

```
0 0
0 1
0 2
0 3
0 4
0 5
1 1
1 2
1 3
1 4
1 5
2 2
2 3
2 4
2 5
3 3
3 4
3 5
4 4
4 5
```

Q5

```
#include<iostream>
using namespace std;
int main() {

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

Output:

```
0 0
0 1
0 2
0 3
0 4
0 5
1 1
1 2
1 3
1 4
1 5
2 2
2 3
2 4
2 5
3 3
3 4
3 5
4 4
4 5
```

Scope: The life-time of a variable. Where does the variable exist and after what line of code will it get destroyed. In short, its accessibility.

Example:

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


int main() {

    int a = 3;
    cout<< a << endl;

    if(true) {
        cout<< a <<endl;
    }


}
```

a is accessible throughout the main() function, after it is declared.

Example:

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


int main() {

    int a = 3;
    cout<< a << endl;

    if(true) {
        int a = 5;
        cout << a <<endl;
    }


}
```

This is 'local' to int main().

This is 'local' to if block, and is only accessible within the if block.

This will print out 3
5

Example:

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


int main() {

    int a = 3;
    cout<< a << endl;

    if(true) {
        int b = 5;
        cout << b <<endl;
    }

    cout<< b << endl;

}
```

??

→ 

Example:

```cpp
int i = 8;
cout << b << endl;

for(; i<8; i++) {
    cout<<" HI " << endl;
}
```

→ Kuch Nahi (Empty)

```
for(; i<8; i++) {
    cout<<" HI " << endl;
}
```

The i outside the for loop gets used.

Precedence Table:

(No need to mug up)

Precedence order	Operator	Associativity
1	() [] →	Left to right
2	++ -- - (unary) ! ~ * & sizeof	Right to left
3	* / %	Left to right
4	+ -	Left to right
5	<< >>	Left to right
6	< <= > >=	Left to right
7	== !=	Left to right
8	& (bitwise AND)	Left to right
9	^ (bitwise XOR)	Left to right
10	| (bitwise OR)	Left to right
11	&& (logical AND)	Left to right
12	|| (logical OR)	Left to right
13	? : (conditional)	Right to right
14	= += -= *= /= %= (assignment operators)	Right to left
15	, (comma Operator)	Left to right

Just like BODMAS prioritizes Brackets, we can also prioritize calculations using Brackets.

Q1. Subtract the product and sum of the digits of an integer.

Soln :

```cpp
class Solution {
public:
    int subtractProductAndSum(int n) {
        int product = 1, sum = 0;
        while(n) {
            product = product * (n % 10); // Can also use *=
            sum += (n % 10);
            n /= 10; // same as n = n / 10;
        }
        return product - sum;
    }
};
```

Success Details >

Runtime: 0 ms, faster than 100.00% of C++ online submissions for Subtract the Product and Sum of Digits of an Integer.

Memory Usage: 5.9 MB, less than 23.16% of C++ online submissions for Subtract the Product and Sum of Digits of an Integer.

Q2. Count number of 1 bits.

Soln :

```cpp
class Solution {
public:
    int hammingWeight(uint32_t n) {
        int ans = 0;
        while(n) {
            if(n & 1) {
                ans++;
            }
            n = n >> 1;
        }
        return ans;
    }
};
```

Q3. Reverse integer.

Soln :

```cpp
class Solution {
public:
    int reverse(int x) {
        int rev = 0;
        while (x != 0) {
            int pop = x % 10;
            x /= 10;
            if (rev > INT_MAX/10 || (rev == INT_MAX / 10 && pop > 7)) return 0;
            if (rev < INT_MIN/10 || (rev == INT_MIN / 10 && pop < -8)) return 0;
            rev = rev * 10 + pop;
        }
        return rev;
    }
};
```

Q4. Complement of Base 10

Soln :

```cpp
class Solution {
public:
    int bitwiseComplement(int n) {
        if(n == 0) return 1;

        int ans = 0, fac = 1;

        while(n != 0){
            int bit = n % 2 == 0;
            ans += fac * bit;
            fac *= 2;
            n /= 2;
        }
        return ans;
    }
};
```

Q5 Number Compliment.

Soln :

```cpp
class Solution {
public:
    int findComplement(int num) {
        int msb = (int)(log2(num));
        if(num ==0) {
            return 1;
        }

        int sum = 0;
        for(int i=msb; i>=0;i--) {
            if(num &(1<<i)){
                continue;
            }
            else {
                sum+=pow(2,i);
            }
        }
        return sum;
    }
};
```

Q6. Binary to Decimal.

Soln :

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

int main()
{
    int num = 1010001;
    int dec_value = 0;
    int base = 1;
    int temp = num;

    while (temp) {
        int last_digit = temp % 10;
        temp = temp / 10;

        dec_value += last_digit * base;

        base = base * 2;
    }
    cout << dec_value << endl;
    return 0;
}
```

Decimal to Binary :

(1) 10 →

$\frac{10}{2} = 5$, $\frac{5}{2} = 2$, $\frac{2}{2} = 1$, $\frac{1}{2} = 0$

(Remainder) 0 1 0 1

Finish ← Start

$2^3\ 2^2\ 2^1\ 2^0$

10 → 1010

Steps : i) Divide number by 2
ii) Store the remainder.
iii) Repeat for the dividend.

Division	Remainder
10/2	0
5/2	1
2/2	0
1/2	1

Reverse = 1010

g. n = 7 to Binary

Division	Remainder
7/2	1
3/2	1
1/2	1

Reverse = 111

(2) n = 5 → Binary ?

Logic : n & 1 = 0 if n is even.

n & 1 = 1 if n is odd

Thus, a bit is 1 if bit & 1 = 1
a bit is 0 if bit & 1 = 0

Thus, (any_bit) & 1 = (any_bit)

∴ Last bit of 5 = 5 & 1 = 1

Check 5 = 101 ⇒

```
  101
& 001
  ---
  (1)
```

Right shift 5 by 1 ⇒ n = 2 (010)

Second last bit of 5 = 2 & 1 = 0

Right shift 2 by 1 ⇒ n = 1 (001)

First bit of 5 = 1 & 1 = 1

Right shift 1 by 1 ⇒ n = 0 (000)

Khatam !

```cpp
#include<iostream>
#include<math.h>
using namespace std;


int main() {

    int n;
    cin >> n;


    int ans  = 0;
    int i = 0;
    while(n != 0 ) {

        int bit  = n & 1;

        ans = (bit * pow(10, i) ) + ans;

        n = n >> 1;
        i++;

    }

    cout<<" Answer is " << ans << endl;


}
```

Example : n = 6

① ans = 0 , i = 0

② bit = 6 & 1 = 0 (even)

③ ans = 10^0 x 0 + ans = 0 , n = 3 , i = 1

④ bit = 3 & 1 = 1 (odd)

⑤ ans = (10^1 x 1) + ans = 10 , n = 1 , i = 2

⑥ bit = 1 & 1 = 1 (odd)

⑦ ans = $(10^2 \times 1)$ + ans = 100 + 10 = 110 , n = 0

ans = 110

Negative Decimal to Binary: (Homework)

-6 ⟶ Binary ??

We have discussed the logic about the storage & display of negative numbers earlier.

$$-6 \rightarrow 6 \rightarrow 000\ldots00110 \rightarrow \begin{array}{r} 111\ldots11001 \\ +\quad\quad\quad\quad 1 \\ \hline 111\ldots11010 \end{array}$$

Think what is 111....11010 equal to if it is unsigned?

$\underbrace{111\ldots11}_{29\text{ bits}}010 = 4294967290$

$= 2^{32} - 6$

$\therefore -6 \xrightarrow{\text{Binary}}$ Binary of $(2^{32} - 6) \rightarrow \underbrace{111\ldots11}_{29}010$

We can't express this in any data type.

Assuming we have 2 byte (= 16 bit) integers:

(Might need online IDE)

```cpp
#include <iostream>
#include <math.h>
using namespace std;

int main(void)
{
    long long int n;
    cin >> n;
    unsigned long long int i = 0, ans = 0;
    if(n < 0) {
        n = pow(2, 16) + n;
    }
    cout << n << endl;
    while(n) {
        int lastBit = n & 1;
        ans = (pow(10, i) * lastBit) + ans;
        n = n >> 1;
        i++;
        cout << ans << endl;
    }
    cout << ans << endl;
    return 0;
}
```

Binary to Decimal:

$101011 = 2^0 \times 1 + 2^1 \times 1 + 2^2 \times 0 +$

$2^5\ 2^4\ 2^3\ 2^2\ 2^1\ 2^0 \quad 2^3 \times 1 + 2^4 \times 0 + 2^5 \times 1$

$= 1 + 2 + 0 + 8 + 0 + 32$

$= \boxed{43}$

Logic: $n \& 1 = \begin{cases} 0, & \text{don't do anything} \\ 1, & \text{multiply with } 2^i \end{cases}$

$i++, \quad n >> 1.$

We will give int n = 10010; but it is actually a decimal number so last bit = n%10;

```
#include<iostream>
#include<math.h>
using namespace std;


int main() {

    int n;
    cin >> n;

    int i = 0, ans = 0 ;

    while( n != 0) {

        int digit = n % 10;

        if( digit == 1) {
            ans = ans + pow(2, i);
        }

        n = n/10;
        i++;

    }
    cout<< ans << endl;


}
```

Q1. Reverse Integer

Example test cases: I/P :- 123
O/P :- 321

I/P :- -123
O/P :- -321

I/P :- -2147483646
O/P :- 0

(Because the reversed integer exceeds the range of int.)

Hint : Check for overflow in every iteration before or after updating ans.

Babbar Code

```
class Solution {
public:
    int reverse(int x) {
        int ans = 0;
        while(x) {
            int digit = x % 10;
            if((ans > INT_MAX/10) || (ans < INT_MIN/10))
                return 0;
            ans = ans * 10 + digit;
            x /= 10;
        }
        return ans;
    }
};
```

Using long long int

```
class Solution {
public:
    int reverse(int x) {
        long long int ans = 0;
        while(x) {
            int lastDigit = x % 10;
            ans = ans*10 + lastDigit;
            if(ans > INT_MAX || ans < INT_MIN)
                return 0;
            x /= 10;
        }
        ans = x < 0 ? -ans : ans;
        return ans;
    }
};
```

Q2 Complement of Base-10 integer.

Approach -1 : n = 5 = 101

$$\wedge \begin{array}{r} 101 \\ 1 \\ \hline 100 \end{array} \quad , \quad \wedge \begin{array}{r} 100 \\ 1 \\ \hline 110 \end{array} \quad , \quad \wedge \begin{array}{r} 110 \\ 1 \\ \hline 010 \end{array} \quad \Rightarrow \quad (2)$$

Let $x = 1$, till $x <= n$, do $n \wedge x$ and then $x << 1$.

```cpp
class Solution {
public:
    int bitwiseComplement(int n) {
        if(n == 0)
            return 1;
        int x = 1;
        while(x <= n) {
            n = n ^ x;
            x = x << 1;
        }
        return n;
    }
};
```

Approach - 2 : Babbar Method. (Video dekho)

```cpp
class Solution {
public:
    int bitwiseComplement(int n) {
        int m = n;
        int mask = 0;
        if(n == 0)
            return 1;

        while(m != 0) {
            mask = (mask << 1) | 1;
            m = m >> 1;
        }

        int ans = (~n) & mask;
        return ans;
    }
};
```

Q3 Power of Two.

Hint : Any negative integer is not a power of 2.

(1)

```cpp
class Solution {
public:
    bool isPowerOfTwo(int n) {

        for(int i = 0; i <= 30; i++) {

            int ans = pow(2,i);

            if(ans == n)
            {
                return true;
            }
        }
        return false;
    }
};
```

→ Brute Force

(2)

Improved Brute Force

```
class Solution {
public:
    bool isPowerOfTwo(int n) {
        int temp = 1;
        while(n != temp && temp < INT_MAX/2) {
            temp*=2;
        }
        return n == temp;
    }
};
```

OR

```
class Solution {
public:
    bool isPowerOfTwo(int n) {

        int ans = 1;

        for(int i = 0; i <= 30; i++) {

            //cout<<" ans "<<ans <<endl;
            if(ans == n)
            {
                return true;
            }
            if(ans < INT_MAX/2)
            ans = ans * 2;

        }
        return false;
    }
};
```

(3) Homework (Most efficient)

```
class Solution {
public:
    bool isPowerOfTwo(int n) {
        if(n <= 0)
            return false;
        return (n & (n - 1)) == 0;
    }
};
```

Hint: n & (n-1) flips the rightmost SET bit.

Eg: n = 18 ⇒ 000....010010

18 & (17) ⇒ 000....010010
& 000....010001
000....010000

Switch Statement and Functions

16 February 2022 15:27

Switch statement is a cleaner way of representing various cases for the same variable as compared to an if-else statement.

If you are repeatedly using if-else if for representing different cases of a variable like :

```
if(   ){

}
else if(   ){

}
else if(   ){

}
else if(   ){

}
:
:
else {

}
```

Then you probably should switch to switch-case statements

SYNTAX:

```
switch(variable_name){

    case value_1 :
                 ~~~~~~~~~~
                 ~~~~~~~~~~~~~
                 break; ← optional
```

```
case value_2 :
    ~~~~~~~~~~~~
    ~~~~~~~~~~~~
    break;  <-- optional.
    .  .  .  .
default :
    ~~~~~~~~~~~~
    ~~~~~~~~~~~~
}
```

⚠ If you didn't understand the syntax, then don't worry and have patience. Everything will be made clear with the code implementation.

A simple usecase of switch-case

```
char ch = '1';

cout << endl;
switch( ch ) {

    case 1: cout << "First" <<endl;
            break;

    case '1': cout << "character one" <<endl;
            break;

    default: cout << " It is default case" <<endl;

}
```

↓ Gives

```
lovebabbar@192 ~ % cd "/Users/lovebabbar/" && g++ switch.
ch && "/Users/lovebabbar/"switch

character one
```

Here, character ch is being checked for cases 1 and '1'.

Clearly, since ch = '1', the second case's condition is fulfilled and we get "character one" as output.

If char ch = 'a', then we will go to the default case if you have made one. Thus, default is optional.

Break :

Using break is optional for each case. If break is not used, all the cases starting from the first case whose condition was fulfilled, will get executed.

If break is used, only that case will get executed whose condition is fulfilled.

→ Without break :

```
char ch = 'a';
int num = 1;

cout << endl;
switch( num ) {

    case 1: cout << "First" <<endl;
            cout << " First again " <<endl;
            //break;

    case '1': cout << "character one" <<endl;
            break;

    default: cout << " It is default case" <<endl;

}
```

We will stop here if case '1' or case 1 is executed.

↓ Gives

```
lovebabbar@192 ~ % cd "/Users/lovebabbar/" && g++ switch.
ch && "/Users/lovebabbar/"switch

First
 First again
character one
```

→ With break :

```
char ch = '1';
/// int num = 1;

cout << endl;
switch( ch ) {

    case 1: cout << "First" <<endl;
            cout << " First again " <<endl;
            break;

    case '1': cout << "character one" <<endl;
            break;

    default: cout << " It is default case" <<endl;

}
```

```
}
```

Gives

```
lovebabbar@192 ~ % cd "/Users/lovebabbar/" && g++ switch.
ch && "/Users/lovebabbar/"switch

character one
```

NESTED SWITCH - CASE EXAMPLE :

```
char ch = '1';
int num = 1;
switch(ch) {
    case 1:
        cout << "First Case" << endl;
        break;
    case '1':
        cout << "Second Case" << endl;
        switch(num) {
            case 0:
                cout << "num = 0" << endl;
                break;
            case 1:
                cout << "num = 1" << endl;
                break;
            case 2:
                cout << "num = 2" << endl;
                break;
            default:
                cout << "don't know what num is" << endl;
                break;
        }
        break;

    default:
        cout << "don't know what ch is" << endl;
}
```

Notice this break. for case `1'

Gives

```
Output.txt
1  Second Case
2  num = 1
3
```

Continue in Switch - Case :

Not possible / valid to use continue statement in switch - case because we are not going through any iterations, thus there is nothing to skip.

Mini - Calculator :

```
int a, b;

cout <<" Enter the value of a "<<endl;
cin >> a;
cout<<" Enter the value of b " <<endl;
cin >> b;
char op;
cout<<" Enter the Operation you want to perform" <<endl;
cin >> op;

switch( op ) {

    case '+':  cout << (a+b) <<endl;
               break;

    case '-': cout<< (a-b) <<endl;
               break;

    case '*': cout<< (a*b) <<endl;
               break;

    case '/': cout<< (a/b) <<endl;
               break;

    case '%': cout<< (a%b) <<endl;
               break;

    default: cout << "Please enter a valid Operation " << endl;

}
```

Homework Solution :

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

int main(void)
{
    int money;
    cin >> money;

    switch (1) {
        case 1:
            cout << "No of 100 Rupee Notes = " << money/100 << endl;
            money = money % 100;

        case 2:
            cout << "No of 50 Rupee Notes = " << money/50 << endl;
            money = money % 50;

        case 3:
            cout << "No of 20 Rupee Notes = " << money/20 << endl;
            money = money % 20;

        case 4:
            cout << "No of 1 Rupee Notes= " << money << endl;
            money = money % 1; // unnecessary

    }
    return 0;
}
```

```
1   1330
```

Output.txt

```
1   No of 100 Rupee Notes = 13
2   No of 50 Rupee Notes = 0
3   No of 20 Rupee Notes = 1
4   No of 1 Rupee Notes= 10
5
```

Functions :

Functions are code blocks / programs that store a code 'template' / 'blueprint' to be used again and again without making the code bulky. Any change that has to be made

is done within the function to reflect it everywhere. Functions make code readable, less bulky, less buggy.

Eg: Find power - 2^5, 3^7, 4^6

① Write the code of power 3 times and each time, take input for a and b. (a^b)

② Write a 'power' function and call it 3 times for different values of a and b as arguments.

```
int power() {
    int a, int b;
    cin >> a >> b;

    int ans = 1;

    for(int i = 1; i<=b; i++) {
        ans = ans * a;
    }

    return ans;
}
```

Calling power(a,b) in main() function.

```
int ans = power();
cout << " answer is " << ans << endl;
```

Output :

```
lovebabbar@192 ~ % cd "/Users/lovebabbar/" && g++ functio
unction1 && "/Users/lovebabbar/"function1
12 2
 answer is 144
```

Check Even :

```
bool isEven(int a) {
    //odd
    if(a&1) {
        return 0;
    }
    else { //Even
        return 1;
    }

}
```

Find ^nC_r :

$$^{n}C_{r} = \frac{n!}{r!\,(n-r)!}$$

We will need another function that finds out factorial of a number for us.

```
int factorial(int n) {

    int fact = 1;

    for(int i = 1; i<=n; i++ ) {
        fact = fact * i;
    }

    return fact;
}

int nCr(int n, int r) {

    int num = factorial(n);

    int denom = factorial(r) * factorial(n-r);

    return num/denom;
}
```

When nCr(5,2) is called in main() function,

```
int num = factorial(5);  // n = 5 for this call

int denom = factorial(2) * factorial(3);  // r = 2

return num / denom;
```

for factorial(5), n = 5 is passed to the function and it returns 120. Similar is the case for n = 2, 3.

Counting Program :

Note : If you don't want to return anything, use void as the return type.

```cpp
void printCounting(int n) {

    for(int i=1; i<=n; i++) {
        cout<< i << " ";
    }
    cout<<endl;
}

int main() {

    int n;
    cin >> n;

    printCounting(n);


    return 0;
}
```

```
lovebabbar@192 ~ % cd "/Users/lovebabbar/" && g++ countin
unting && "/Users/lovebabbar/"counting
10
1 2 3 4 5 6 7 8 9 10
```

Check Prime :

Return type ←
void/int/float/char/bool

```cpp
// 1 -> Prime no.
// 0 -> Not a Prime no.
bool isPrime(int n) {

    for( int i = 2; i<n; i++) {
        //divide hogya h , not a prime no.
        if(n%i == 0) {
            return 0;
        }
    }

    return 1;
}
```

function parameters : variables local to this function that will receive some value from the fun. where it's called.

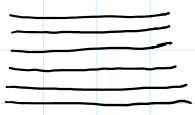
return statements

Function Call Stack :

Stack is a Last - In - First - Out `thing'. Example : Stack of plates.

Put new plates on top

Take the top plate / last plate away first.

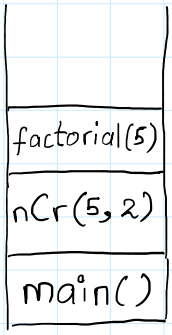


Last In

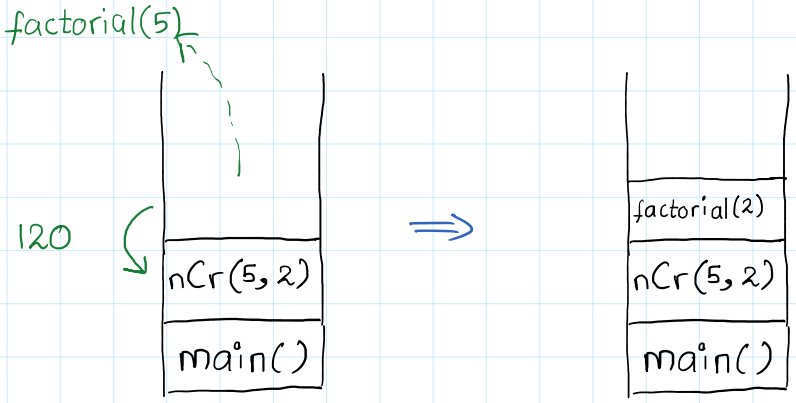
First Out

Functions that are called, get stored in a stack like structure.

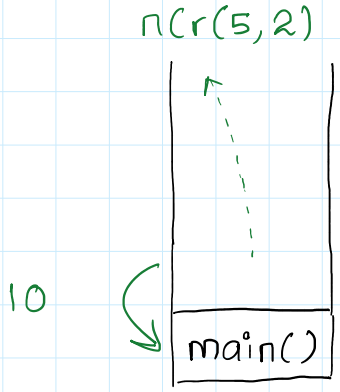
Eg: main() Call → nCr(5,2) Call → factorial(5)



factorial(5) returns 120 and then factorial(2) is called.



and so on.... till nCr(5,2) returns 10.



Homework Questions :

Q1. Print 3*n + 7 A.P.

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

int ap(int n) {
    return n*3 + 7;
}

int main(void)
{
    int n;
    cin >> n;
    cout << ap(n) << endl;
    return 0;
}
```

```
1  3
```

Output.txt

```
1  16
```

Q2. Print total no of SET bits in A & B.

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

int setBits(int num) {
    int ans = 0;
    while(num) {
        num = num & (num-1);
        ans++;
    }
    return ans;
}

int main(void)
{
    int a, b;
    cin >> a >> b;
    cout << setBits(a) + setBits(b) << endl;
    return 0;
}
```

```
1  11
2  2
```

Output.txt

```
1  4
2
```

Q3. Fibonacci Number

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

int fibonacciNumber(int n) {
    if(n == 1 || n == 2)
        return n-1;
    int ans = 1, a = 0, b = 1;
    for(int i=3;i<=n;i++) {
        ans = a + b;
        a = b;
        b = ans;
    }
    return ans;
}

int main(void)
{
    int n;
    cin >> n;
    cout << fibonacciNumber(n) << endl;
    return 0;
}
```

```
1  8
```

Output.txt

```
1  13
2
```

Call By Value :

When we pass a variable (not its pointer / address) to a function, we pass its value instead of passing the entire

variable (original variable). This is call by value where we pass the copy of the variable instead of the original variable present at some memory address.

If we call fibonacciNumber(6), then we are just giving the value 6 in place of n everywhere in the funct.

```
int fibonacciNumber(int n) {
    if(n == 1 || n == 2)
        return n-1;
    int ans = 1, a = 0, b = 1;
    for(int i=3;i<=n;i++) {
        ans = a + b;
        a = b;
        b = ans;
    }
    return ans;
}
```

If in main() function,

```
int n = 6;
cout << fibonacciNumber(n);
```

And change the value of n in fibonacciNumber(), then we will not change the value of the original n in the main() function.

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

void fun(int n) {
    cout << "value of copy in fun() before changing = " << n << endl;
    n += 10;
    cout << "value of copy in fun() after changing = " << n << endl;
}

int main(void)
{
    int n;
    cin >> n;
    cout << "value of real n before function call = " << n << endl;
    fun(n);
    cout << "value of real n after function call = " << n << endl;

    return 0;
}
```

```
1  10
```

Output.txt

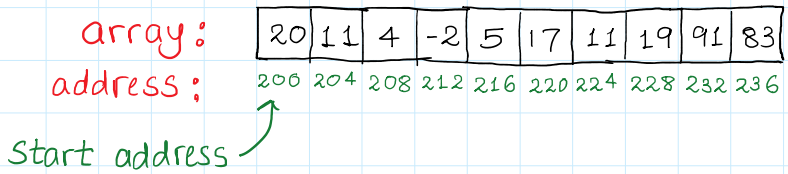
```
1  value of real n before function call = 10
2  value of copy in fun() before changing = 10
3  value of copy in fun() after changing = 20
4  value of real n after function call = 10
5
```

Arrays

18 February 2022 13:12

Arrays help you store variables of the same data type at contiguous memory locations.

Store 10 integers. (Suppose int takes 4 bytes)

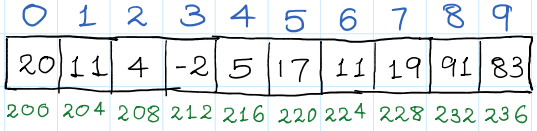


The contiguous memory blocks help us when we try accessing the elements of the array.

Given the start address (address of the array in memory), I can access all the elements by simply incrementing the address by 4 (size of data type).

Suppose our array is named arr. We represent an array as arr[].

To access the elements of an array, we use indices (plural of index). We follow a 0-based indexing in most programming languages



n = 10

∴ arr[0] = 20, arr[6] = 11

Thus, arrays are a 'data structure' that help storing multiple variables of the same data type in contiguous memory locations

DECLARATION OF ARRAYS:

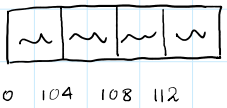
1. Declaring an array to store 100 integers:

```
int arr[100];
```

2. Declaring an array to store 100 booleans:

```
bool isGood[100];
```

Now, in examples ① and ②, arr and isGood are the addresses of the respective arrays and point to the memory location where the first element is stored.

Eg: `int arr[4];` → 

~ : garbage value

where arr[0] is 100 – 103 (4 bytes)
arr[1] is 104 – 107 (4 bytes)
arr[2] is 108 – 111 (4 bytes)
arr[3] is 112 – 115 (4 bytes)

If you print arr, you will get 100 in output.

```
cout << arr;
```
STDOUT: 100

Which is the address of the first element arr[0].

```
cout << &arr[0];
```
STDOUT: 100.

```
cout << arr[1];      // Suppose arr[]:
```

0	1	2	3
4	8	11	1

STDOUT: 8

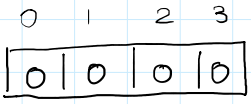
INITIALIZATION:

① `int num[5] = {5,18,40,12}`

0	1	2	3
5	18	40	12

② Initializing an array with 0.

```
int arr[4] = {0};
```



③ Initializing an array with any number. (Homework)

```
int arr[10];
std::fill(arr, arr+10, 27);   // std:: is optional if you have declared the namespace as discussed in previous lectures.
```

start address — end address — value

OR

```
int arr[10];
std::fill_n(arr, 10, 27);
```

start address — No. of elements from the start address to fill — Value

ACCESSING ELEMENTS OUT OF RANGE:

```
//declare
int number[15];

//accessing an array
cout << "Value at 14 index " << number[14] << endl;

cout << "Value at 20 index " << number[20] << endl;
```

⚠ Accessing index out of range will throw an error.

Here, you can only access elements from number[0] to number[14].

```
lovebabbar@192 ~ % cd "/Users/lovebabbar/" && g++ arraysIntro.cpp -o
 arraysIntro && "/Users/lovebabbar/"arraysIntro
arraysIntro.cpp:17:37: warning: array index 20 is past the end of t
e array (which contains 15 elements) [-Warray-bounds]
    cout << "Value at 20 index " << number[20] << endl;
                                    ^      ~~
arraysIntro.cpp:12:5: note: array 'number' declared here
    int number[15];
    ^
1 warning generated.
Value at 14 index 1
Value at 20 index 1
```

PRINTING AN ARRAY:

```
//print the array
for(int i =0; i<n; i++) {
    cout << third[i] << " ";
}
```

where n is the size of the array. See here that our loop is running from 0 to n-1 (not n). Name of our array is third.

Printing array using a function:

```
void printArray(int arr[], int size) {

    cout<< " printing the array " << endl;
    //print the array
    for(int i =0; i<size;i++) {
        cout << arr[i] << " ";
    }
    cout<< " printing DONE " << endl;
}
```

We have to pass the size because int arr[] in the function declaration is simply the address of the array and gives us no information about the size.
Also, if the array has space for 1000 elements but I am storing just 6, then I can pass size as 6 to the function.

MAXIMUM / MINIMUM IN AN ARRAY:

Any value we enter in our array would be less than or equal to max.

```
int getMax(int num[], int n) {

    int max = INT_MIN;

    for(int i = 0; i<n; i++) {
        if(num[i] > max){
            max = num[i];
        }
    }

    //returning max value
    return max;
}
```

```
int getMin(int num[], int n) {

    int min = INT_MAX;

    for(int i = 0; i<n; i++) {
        if(num[i] < min){
            min = num[i];
        }
    }

    //returning min value
    return min;
}
```

```
int main() {

    int size;
    cin >> size;

    int num[100];

    //taking input in array
    for(int i = 0; i<size; i++) {
        cin >> num[i];
    }

    cout << " Maximum value is " << getMax(num, size) << endl;
    cout << " Minimum value is " << getMin(num, size) << endl;


    return 0;
}
```

CAUTION: Never declare a variable-sized array. Always specify a constant size while declaring your array even if it turns out to be very large as compared to your required array size.

ARRAYS ARE ALWAYS PASSED BY REFERENCE:

When we pass an array to a function like this:

func (int arr[]) → address of arr, say 100.

We are not passing a copy of the array unlike variables, since array name 'arr' is the address of the array, thus we pass the address of the array to the function allowing it to make changes to the original array.

This is called passing by reference of array.

HOMEWORK: Print sum of all elements of an array.

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

int getSumOfArray(int arr[], int n) {
    int sum = 0;
    for(int i=0;i<n;i++) {
        sum += arr[i];
    }
    return sum;
}

int main(void)
{
    int n;
    cin >> n;
    int arr[1000];
    for(int i=0;i<n;i++) {
        cin >> arr[i];
    }
    int sum = getSumOfArray(arr, n);
    cout << "Sum of all elements of the array = " << sum << endl;
    return 0;
}
```

```
1  5
2  -4 9 0 11 2
```

Output.txt

```
1  Sum of all elements of the array = 18
2
```

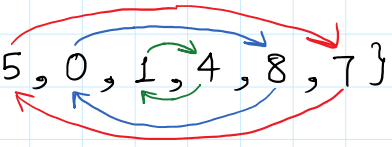
LINEAR SEARCH :

```cpp
bool search(int arr[], int size, int key) {

    for( int i = 0; i<size; i++ ) {

        if( arr[i] == key) {
            return 1;
        }

    }
    return 0;
}
```

Traversing the array and comparing each element with the search key.

REVERSE AN ARRAY :

arr : {5, 0, 1, 4, 8, 7}



{7, 8, 4, 1, 0, 5}

```cpp
void reverse(int arr[], int n) {

    int start = 0;
    int end = n-1;

    while(start<=end) {
        swap(arr[start], arr[end]);
        start++;
        end--;
    }
}
```

Q1. Swap Alternate.

Approach: Given an array, say [1,2,3,4,5], we will run a for loop from 0 to n-2 and swap arr[i] with arr[i+1]. If we reach the last element, we will stop, so check if $(i+1)^{th}$ element < size of array.

Dry Run: arr[] : {1,2,3,4,5}

① {1,2,3,4,5} ⇒ {2,1,3,4,5}

② {2,1,3,4,5} ⇒ {2,1,4,3,5}

③ {2,1,4,3,5} Stop

Code:

```
void swapAlternate(int arr[], int size) {

    for(int i = 0; i<size; i+=2 ) {
        if(i+1 < size) {
            swap(arr[i], arr[i+1]);
        }
    }

}
```

SWAP FUNCTION:

If a = 5, b = 10, I can't simply do a = b, b = a to swap them. Suppose I did this:

```
int a = 5, b = 10;
a = b; // a becomes 10
b = a; // b becomes 10
```

a = 10 b = 10

Use a third variable to temporarily store the value of a and then change a to b.

```
int a = 5, b = 10;
int temp = a; // temp = 5.
a = b; // a becomes 10
b = temp; // b becomes 5.
```

a = 10 b = 5

```
void swap(int *a, int *b) {
    // pointers to numbers are given
    int temp = *a; // Store the value at address a
    *a = *b; // Change the value at address a to the value at address b
    *b = temp; // Change the value at address b to the original value of a
}
```

Q2. Find Unique Element.

Approach: Since all numbers except any one occur twice, thus if there are m numbers that occur twice and one that occurs once, I will have 2m+1 numbers.

The XOR of same numbers is 0. Thus 5^1^5 = 1.

We can find XOR of all numbers, which will be the only element occuring once.

Dry Run: arr[] : {1, 2, 8, 2, 1, 7, 7}

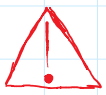
1^2^8^2^1^7^7 = 0^8 = 8

Code:

```
int findUnique(int *arr, int size)
{
    int ans = 0;

    for(int i = 0; i<size; i++) {
        ans = ans^arr[i];
    }
    return ans;
}
```

Q3 Unique Number of Occurances. (Homework)

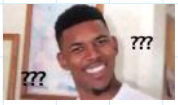
We can use an unordered map that uses the concept of



We can use an unordered map that uses the concept of hashing. Learn about it and then use it by yourself.

```cpp
bool uniqueOccurrences(vector<int>& arr) {
    vector<int> occurance(2001, 0);
    // ith index is occurance of (i-1000)

    for(int i=0; i<arr.size(); i++) {
        occurance[arr[i]+1000]++;
    }
    // check if all non-zero elements are unique
    sort(occurance.begin(), occurance.end());
    // if any adjacenet non-zero neighbours are equal, then we have repeating occurances
    for(int i=0;i<2000;i++) {
        if(occurance[i] != 0 && occurance[i] == occurance[i+1])
            return false;
    }
    return true;
}
```

```cpp
bool uniqueOccurrences(vector<int>& arr) {
    unordered_map<int, int> m;
    for(int i=0;i<arr.size();i++) {
        m[arr[i]]++;
    }
    // insert the occurances into a set that will store them in increasing order
    set<int> occ;
    for(auto x: m) {
        occ.insert(x.second);
    }
    // if the set has the same number of elements as the map then there were all unique occurances
    return occ.size() == m.size();
}
```



Q4 Find Duplicates

Approach 1 : XOR all the elements of the array with $1, 2, 3, 4, \ldots, n-1$.

Approach 2 : Subtract the sum of first n-1 natural nos. from the sum of all elements of the array.

Dry Run : arr[] : {4, 2, 1, 3, 1}

n-1 Natural Numbers : 1, 2, 3, 4

XOR : $(4 \wedge 2 \wedge 1 \wedge 3 \wedge 1) \wedge (1 \wedge 2 \wedge 3 \wedge 4) = \boxed{1}$

sum(arr) = 4 + 2 + 1 + 3 + 1 = 11. — (1)

sum of (n-1) natural nos. = 1 + 2 + 3 + 4 = 10 — (2)

$\left(\frac{(n-1)*n}{2}\right)$

eq.(1) - eq.(2) :

$$
\begin{array}{r}
4+2+1+3+1 \\
-\ 1+2+3+4 \\
\hline
1 \\
\hline
\end{array}
$$

Code:

```
int findDuplicate(vector<int> &arr)
{
    int ans = 0;

    for(int i = 0; i<arr.size(); i++ ) {
        ans = ans^arr[i];
    }

    for(int i = 1; i<arr.size();i++ ) {
        ans = ans^i;
    }
    return ans;
}
```

Homework: Find all duplicates in an array.

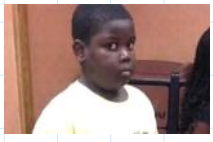
Approaches:

1. For every element, check if there exists a duplicate element in arr[i+1....n-1], if yes then store it in answer array.

2. Sort the array and check for adjacent elements if they are equal.

3. Track frequency using an unordered map (Hash Map) and traverse it to get all the repeating elements.

4. Treat the element at arr[i] as the next index to jump on and mark this new element as negative to show that you have already been to arr[i].
   If you visit an element and found it negative, then you have visited it twice.



Most Efficient Approach's Dry Run:

```
          i
arr[]: {4, 3, 2, 5, 2}
       0  1  2  3  4
```

arr[i] = 4 for i = 0, visit arr[4-1] = arr[3] = 5

and mark it -ve

i

{4, 3, 2, -5, 3}

arr[i] = 3 for i = 1, visit arr[3-1] = arr[2] = 2
and mark it -ve.

{4, 3, -2, -5, 3}

arr[i] = 2 for i = 2, visit arr[2-1] = arr[1] = 3
and mark it -ve.

{4, -3, -2, -5, 3}

arr[i] = 5 for i = 3, visit arr[5-1] = arr[4] = 3
and mark it -ve.

{4, 3, -2, -5, -3}

arr[i] = 3 for i = 4, visit arr[3-1] = arr[2] = -2
Since it's already negative, we have visited arr[2] twice and thus 3 has been used twice to check for arr[3-1]

∴ 3 is occuring twice.

Code :

```
vector<int> findDuplicates(vector<int>& nums) {
    vector<int> ans;
    for(int i=0; i<nums.size(); i++) {
        // visit nums[abs(nums[i]) - 1]
        // abs is used because the current element might be negative
        if(nums[abs(nums[i]) - 1] < 0) {
            ans.push_back(abs(nums[i]));
        }
        else {
            nums[abs(nums[i]) - 1] = -nums[abs(nums[i]) - 1];
        }
    }
    return ans;
}
```

Q5 Find Intersection.

Approach : Maintain two trackers i and j for the first and second array respectively.
At any point of time if :

① arr[i] < arr[j] → increment i
② arr[i] > arr[j] → increment j
③ arr[i] == arr[j] → include in answer and increment both i and j.

Code:

```
vector<int> findArrayIntersection(vector<int> &arr1, int n, vector<int> &arr2, int m)
{
    vector<int> ans;
    int i = 0, j = 0;
    while(i < arr1.size() && j < arr2.size()) {
        if(arr1[i] < arr2[j]) {
            i++;
        }
        else if(arr1[i] > arr2[j]) {
            j++;
        }
        else {
            ans.push_back(arr1[i]);
            i++;
            j++;
        }
    }
    return ans;
}
```

Q6. Pair Sum.

Approach: For the i^{th} element, check if its sum with all $i+1, i+2, \ldots, n-1$ elements == sum. If no then continue, if yes then push the i^{th} & j^{th} elements in an answer vector.

Dry Run: arr[] : {1, 2, 3, 4}
sum = 5.

① arr[] : {1, 2, 3, 4} (i)
Check arr[j] for all j = 1 to 3.
arr[i] = 1 , 1 + arr[j] == 5.
This is satisfied for j = 3, 1 + 4 = 5.
⇒ ans.push_back({1,4})

② arr[] : {1, 2, 3, 4} (i)
Check arr[j] for all j = 2 to 3.
arr[i] = 2 , 2 + arr[j] == 5.
This is satisfied for j = 2, 2 + 3 = 5.
→

⇒ ans.push_back({2,3})

③ arr[] : {1, 2, 3, 4}

Check arr[j] for all j = 3 to 3.

arr[i] = 3 , 3 + arr[j] == 5.

This is not satisfied for any pair.

Code :

```
vector<vector<int> > pairSum(vector<int> &arr, int s){
   vector< vector<int> > ans;

   for(int i=0;i<arr.size(); i++ )
   {
              for(int j = i+1; j<arr.size(); j++) {
              if(arr[i] +arr[j] == s)
          {
              vector<int> temp;
              temp.push_back(min(arr[i], arr[j]));
              temp.push_back(max(arr[i], arr[j]));
                            ans.push_back(temp);
          }
       }
   }
   sort(ans.begin(), ans.end());
   return ans;
}
```

Homework : Find triplet sum.

Same Approach as Find Pair Sum.

Code :

```
#include <set>
vector<vector<int>> findTriplets(vector<int>arr, int n, int K) {
    vector<vector<int>> ans;
    set<vector<int>> s;
    for(int i=0;i<n;i++) {
        for(int j = i + 1; j < n; j++) {
            for(int k = j + 1; k < n; k++) {
                if(arr[i]+arr[j]+arr[k] == K) {
                    vector<int> temp;
                    temp.push_back(arr[i]);
                    temp.push_back(arr[j]);
                    temp.push_back(arr[k]);
                    sort(temp.begin(), temp.end());
                    s.insert(temp);
                }
            }
        }
    }
    for(auto x: s) {
        ans.push_back(x);
    }
    return ans;
}
```

Q7. Sort 0, 1

Approach: ① Traverse array & update count of 0 and 1. Again, traverse the array and fill 0s for the first numZero indices and 1 for the remaining indices.

② Maintain 2 pointing variables i and j, i=0 & j=n-1. In any condition, if:

i) arr[i] == 0 → i++
ii) arr[j] == 1 → j--
iii) arr[i] == 1 && arr[j] == 0 → swap them
i++, j--

Dry Run: arr[]: {1,0,0,1,1,1,0,1}

{1,0,0,1,1,1,0,1} → decrement j;
{1,0,0,1,1,1,0,1} → swap(arr[i++], arr[j--]);
{1,0,0,1,1,1,0,1} → increment i;
{1,0,0,1,1,1,0,1} → increment i;
{1,0,0,1,1,1,0,1} → decrement j;
{1,0,0,1,1,1,0,1} → decrement j;
{1,0,0,1,1,1,0,1} →

Code :

```
void printArr(int arr[], int n) {
    for(int i=0;i<n;i++) {
        cout << arr[i] << " ";
    }
    cout << endl;
}

void sortOneTwo(int arr[], int n) {
    int i = 0, j = n-1;
    while(i < j) {
        if(arr[i] == 0) {
            i++;
        }
        else if(arr[j] == 1) {
            j--;
        }
        else if(arr[i] == 1 && arr[j] == 0) {
            swap(arr[i], arr[j]);
            i++;
            j--;
        }
    }
}
```

```
8
1 1 0 0 0 0 1 0

Output.txt
0 0 0 0 0 1 1 1
```

Homework : Sort 0 1 2.

Approach : Take 3 pointing variables i, j and k where i, j and k are such that :

arr[0.... i-1] = 0
arr[k+1 .... n-1] = 2

i = 0, j = 0 and k = n-1.

arr[] : {0, 1, 1, 0, 2, 1, 0}
i j k

If arr[j] == 0, swap(arr[i], arr[j]) and then i++, j++.

If arr[j] == 1, increment j.

If arr[j] == 2, swap(arr[j], arr[k]) and then k--

{0, 1, 1, 0, 2, 1, 0} → swapping arr[i], arr[j] & i++, j++.
i j k

{0, 1, 1, 0, 2, 1, 0} → j++
i j k

{0, 1, 1, 0, 2, 1, 0} → j++
i j k

{0, 1, 1, 0, 2, 1, 0} → j++
 i j   K

{0, 1, 1, 0, 2, 1, 0} → swapping arr[i], arr[j] & i++, j++.
 i j   K

{0, 0, 1, 1, 2, 1, 0} → Swapping arr[j], arr[K] & K--
    i j   K

{0, 0, 1, 1, 0, 1, 2} → swapping arr[i], arr[j] & i++, j++.
    i j K

{0, 0, 0, 1, 1, 1, 2} → j++
       i j K

{0, 0, 0, 1, 1, 1, 2} →
       i K j

Observe how we are mainting arr[0....i-1] = 0 and

arr[K+1 ... n-1] = 2.

Why are we not incrementing j when swapping (arr[j], arr[K]) ?

arr[K] might be 0. If I swap arr[j] with arr[K], then we will get arr[j] = 0, we want swap this with arr[i] in the next step so we should not omit arr[j] by incrementing j.

Code :

```
void sort012(int *arr, int n)
{
    int i = 0, j = 0, k = n-1;
    while(j <= k) {
        if(arr[j] == 1)
            j++;
        else if(arr[j] == 0) {
            swap(arr[i++], arr[j++]);
        }
        else {
            swap(arr[j], arr[k--]);
        }
    }
}
```

Time and Space Complexity

21 February 2022 20:38

What is time complexity ?

Ans : Time complexity gives the "idea" of the amount of time taken by an algorithm as a function of the input size.

It does not necessarily depict the exact execution-time taken by an algorithm, because run-time of a program depends on the following factors :

1. Computer Hardware / Architecture
2. Background Processes and load on the system during execution.
3. The exact location in memory where the program is stored.

, etc.

Thus, it's impossible to determine the execution time of a program that is applicable in all possible cases.

To compare different algorithms, we can easily calculate the time (and space) complexity of a program and judge which is faster and in general, more optimized.

How to Calculate ?

Big Oh is used to denote the worst-case time complexity (upper-bound) of an algorithm.

Eg : I take at most 15 minutes to finish my lunch.

Constant Time : $O(1)$
Linear Time : $O(n)$.

Eg:
```
for(int i=0; i<10; i++){
```

```
        cout << "Hello" << endl;
    }
```

This will always run the loop 10 times irrespective of any input given. Thus, constant time complexity – $O(1)$.

Eg:
```
for(int i=0; i<n; i++){
    cout << "Hello" << endl;
}
```

This will always run the loop n times which depends on n 'linearly'. Thus, linear time complexity – $O(n)$.

Order of Time Complexity :

$$O(1) < O(\log n) < O(\sqrt{n}) < O(n) < O(n \log n) < O(n^2) < O(n^3) < O(2^n) < O(n!)$$

← Faster / Efficient

Finding Upper Bound :

Q1: $f(n) = 4n^4 + 3n^2 + 2$

1. Remove all terms of lower power/precendence (refer to the order above).
2. Remove constants multiplied/divided with / from the highest precendence term.

1. $O(f(n)) = 4n^4 + 3n^2 + 2 \approx 4n^4$
2. $O(f(n)) = 4 \cdot n^4 = \boxed{n^4}$

$\Rightarrow$ $f(n) = O(n^4)$ which means that $f(n)$ will always be upper-bounded/smaller than n^4 after some value of $n = n_0$.

Q2 $f(n) = 4\log n + 5\sqrt{n} + 17$

$O(f(n)) = 4\log n + 5\sqrt{n} + 17 = \boxed{\sqrt{n}}$

$f(n) = O(\sqrt{n})$

Q3 Reverse an array.

→ We swap arr[i] with arr[j] starting with i = 0 & j = n-1 and increment & decrement i & j respectively until i < j.

Thus, we make approx. $\frac{n}{2}$ swaps for an array of length n.

$f(n) = \frac{n}{2} \Rightarrow O(f(n)) = n$

$f(n) = O(n)$

Q4 Linear Search.

We traverse the whole array, so :

$f(n) = n \Rightarrow O(f(n)) = n$

$f(n) = O(n)$

Q5.

```
int a = 0, b = 0;
for (i = 0; i < N; i++) {
    for (j = 0; j < N; j++) {
        a = a + j;
    }
}
for (k = 0; k < N; k++) {
    b = b + k;
}
```

} Runs N times for each iteration i : 0 to N-1

} Runs N times.

Time Complexity = $O(f(N)) = O(N^2 + N) = O(N^2)$

Q6.

```
int a = 0;
for (i = 0; i < N; i++) {
```

Careful !

Q6.

```
int a = 0;
for (i = 0; i < N; i++) {
    for (j = N; j > i; j--) {
        a = a + i + j;
    }
}
```

Careful!

Runs from N to i+1 times for i: 0 to N-1.

Detailed Soln:

For i = 0 : Inner for loop runs N times (j=N; j>0; j--)

For i = 1 : Inner for loop runs N-1 times (j=N; j>1; j--)

For i = 1 : Inner for loop runs N-2 times (j=N; j>2; j--)

⋮ ⋮

For i = N-1: Inner for loop runs 1 time.

Total: $N + (N-1) + (N-2) + \ldots + 1$

$= 1 + 2 + 3 + \ldots + N$

$= \frac{N(N+1)}{2}$

$= \frac{N^2}{2} + \frac{N}{2}$

$f(n) = \frac{N^2}{2} + \frac{N}{2} = \boxed{O(N^2)}$

10^8 Operation Rule:

Most modern machines can perform 10^8 operations/second.

We use the constraints given in a question to determine the maximum T.C. I can have in my solution.

input size	required time complexity
$n \leq 10$	$O(n!)$
$n \leq 20$	$O(2^n)$
$n \leq 500$	$O(n^3)$
$n \leq 5000$	$O(n^2)$
$n \leq 10^6$	$O(n \log n)$ or $O(n)$
n is large	$O(1)$ or $O(\log n)$

Source : CodeForces.

Space Complexity : Gives the 'idea' of the amount of space required by a program " with respect to " the input.

O(1) Space Complexity -

1. `int a;`
2. `int a, b, c, d, z;`
3. `int arr[1000];`

O(n) Space Complexity :

1. `int arr[n];` // Bad practice
2. `vector<int> v(n);`

Binary Search
22 February 2022 19:35

Binary Search is a searching algorithm that works in O(logn) time complexity.

IMPORTANT :
Binary Seach is only applicable for 'monotonic search spaces'. For example, B.S. can be applied for searching an element in a SORTED ARRAY only.

A sorted array where elements are arranged in a non-decreasing order is a monotonically increasing search space. Similarly, if elements are arranged in a non-increasing order, then it's a monotonically decreasing search space.

Eg : arr[] : {1, 3, 3, 6, 7, 10} → monotonically increasing search space

APPLYING BINARY SEARCH :

arr[] : {1, 3, 3, 6, 7, 10} search = 7

(1) Fix a low and high index for the array where you want to search.

arr[] : {1, 3, 3, 6, 7, 10}
low high

(2) Find the mid index for the given low & high index.

mid = (low + high)/2; → (0+5)/2 = 2

0 1 2 3 4 5
arr[] : {1, 3, 3, 6, 7, 10}
low mid high

(3) Check the following :

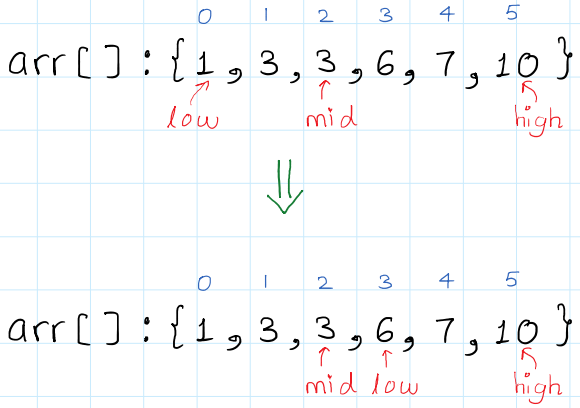
(i) If arr[mid] == search, return mid, you found it!

(ii) If arr[mid] < search, then all elements arr[0....mid] will definitely be smaller than search. Thus, update your search space : low = mid + 1.

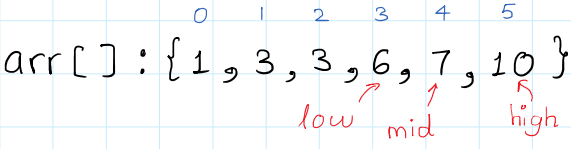
(You have discarded 0.... mid indices)

(iii) If arr[mid] > search, then all elements arr[mid.... n-1] will definitely be greater than search. Thus, update your search space : high = mid - 1.

(You have discarded mid.... n-1 indices)



④ Repeat steps ② and ③ untill low <= high.



arr[mid] == 7, return mid;

return 4;

EXAMPLE : arr[] : {4, 8, 16, 32, 64} K = 4

① {4, 8, 16, 32, 64}
low mid high

② (arr[mid] = 16) > 4 ⇒ high = mid - 1.

{4, 8, 16, 32, 64}
low, high, mid

③ (arr[mid] = 8) > 4 ⇒ high = mid - 1

mid
{4, 8, 16, 32, 64}
low, high

④ (arr[mid] = 4) We found it !

Note: If low > high and we haven't found the element, then the element doesn't exist at all.

Code:

```
int binarySearch(int arr[], int size, int key) {

    int start = 0;
    int end = size-1;

    int mid = (start + end)/2;

    while(start <= end) {

        if(arr[mid] == key) {
            return mid;
        }

        //go to right wala part
        if(key > arr[mid]) {
            start = mid + 1;
        }
        else{
            end = mid - 1;
        }

        mid = (start+end)/2;
    }
    return -1;
}
```

```cpp
int even[6] = {2,4,6,8,12,18};
int odd[5] = {3, 8, 11, 14, 16};

int evenIndex = binarySearch(even, 6, 20);

cout << " Index of 18 is " << evenIndex << endl;

int oddIndex = binarySearch(odd, 5, 8);

cout << " Index of 8 is " << oddIndex << endl;
```

```
lovebabbar@192 ~ % cd "/Users/lovebabbar/" && g++ binarySearch.cpp -
o binarySearch && "/Users/lovebabbar/"binarySearch
 Index of 18 is -1
 Index of 8 is 1
```

CAUTION :

What if you had an array of size $2^{31} - 1$. and during some iteration :

low = $2^{31} - 2$ and high = $2^{31} - 1$,

$$mid = \frac{(low + high)}{2}$$

will throw integer overflow error because $2^{31} + 2^{31} - 1 = 2^{32} - 1$

which is greater than INT_MAX. Thus, we can either use long long or even better,

$$mid = low + \frac{(high - low)}{2}$$

New Code :

```cpp
int binarySearch(int arr[], int size, int key) {

    int start = 0;
    int end = size-1;

    int mid = start + (end-start)/2;

    while(start <= end) {

        if(arr[mid] == key) {
            return mid;
        }

        //go to right wala part
        if(key > arr[mid]) {
            start = mid + 1;
        }
        else{ //key < arr[mid]
            end = mid - 1;
        }

        mid = start + (end-start)/2;
    }
    return -1;
}
```

Linear Search v/s Binary Search :

Linear Search's Time Complexity = $O(n)$.

In Binary Search, we are essentially halving our search space in every iteration.

$\Rightarrow n \rightarrow \frac{n}{2} \rightarrow \frac{n}{4} \rightarrow \frac{n}{8} \rightarrow \dots \rightarrow 1.$

Applying Geometric Progression :

$a = n$
$r = 1/2$

we want to know the no. of elements passed by the time we reach 1.

Say 1 is the z^{th} element.

$ar^{z-1} = 1$
$\rightarrow n\left(\frac{1}{2}\right)^{z-1} = 1$
$\rightarrow 2^{z-1} = n$
$\rightarrow 2^{z} = 2n$
$\rightarrow$ Taking $\log_2$ on both sides :

$\log 2^{z} = \log 2n$

$\rightarrow z = \log(2n) \approx \boxed{O(\log n)}$

Q1. Find the first and last occurance of an element in a sorted array.

Approach: Finding First Occurance.

→ Applying binary search, if at any instance mid is such that arr[mid] == key, then:

① Either this is the first occurance itself.
② Or there are more 'key' towards the left of mid.

So we'll temporarily store mid and then search in the left half by making high = mid - 1.

If arr[mid] < key, then search in right half. by making low = mid + 1.

If arr[mid] > key, then search in left half. by making high = mid - 1.

Example: arr[] = {1, 1, 2, 3, 4, 4, 4, 4} key = 4.

① arr[] = {1, 1, 2, 3, 4, 4, 4, 4}
l mid h

arr[mid] < key ⇒ Definitely in the right half.

low = mid + 1. // low = 4

② arr[] = {1, 1, 2, 3, 4, 4, 4, 4}
l mid h

arr[mid] == key ⇒ Either mid or before mid.
store mid in a variable ans. // ans = 5.
Then make high = mid - 1. // high = 4

mid
③ arr[] = {1, 1, 2, 3, 4, 4, 4, 4}

③ arr[] = {1, 1, 2, 3, 4, 4, 4, 4}

mid, l, h

arr[mid] == key ⇒ Either mid or before mid.
store mid in a variable ans. // ans = 4
Then make high = mid - 1. // high = 3

④ arr[] = {1, 1, 2, 3, 4, 4, 4, 4}

h, l

low > high ⇒ break out. Last value of **ans = 4**

Note: low, mid, high and ans are storing the indices and not values.

Finding Last Occurance.

→ Applying binary search, if at any instance mid is such that arr[mid] == key, then:

① Either this is the last occurance itself.
② Or there are more 'key' towards the right of mid.

So we'll temporarily store mid and then search in the right half by making low = mid+1.

If arr[mid] < key, then search in right half. by making low = mid + 1.

If arr[mid] > key, then search in left half. by making high= mid - 1.

Example: arr[] = {1,1,2,3,4,4,4,4} key = 4.

① arr[] = {1, 1, 2, 3, 4, 4, 4, 4}

l, mid, h

arr[mid] < key ⇒ Definitely in the right half.

low = mid + 1. // low = 4

② arr[] = {1, 1, 2, 3, 4, 4, 4, 4}
l mid h

arr[mid] == key ⇒ Either mid or after mid.
store mid in a variable ans. // ans = 5.
Then make low = mid + 1. // low = 6

③ arr[] = {1, 1, 2, 3, 4, 4, 4, 4}
mid
l h

arr[mid] == key ⇒ Either mid or after mid.
store mid in a variable ans. // ans = 6
Then make low = mid + 1. // low = 7

④ arr[] = {1, 1, 2, 3, 4, 4, 4, 4}
mid
l h

arr[mid] == key ⇒ Either the last occurance is at index mid or to the right of mid.
Store mid in a variable ans. // ans = 7
Then make low = mid + 1.

⑤ arr[] = {1, 1, 2, 3, 4, 4, 4, 4}
h l

low > high ⇒ break out. Last value of ans = 7

Code :

```
int firstOcc(int arr[], int n, int key) {

    int s = 0, e = n-1;
    int mid = s + (e-s)/2;
    int ans = -1;
    while(s<=e) {

        if(arr[mid] == key){
            ans = mid;
            e = mid - 1;
        }
        else if(key > arr[mid]) {//Right me jao
            s = mid + 1;
        }
        else if(key < arr[mid]) {//left me jao
            e = mid - 1;
        }

        mid = s + (e-s)/2;
    }
    return ans;
}
```

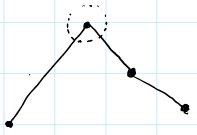
```
int lastOcc(int arr[], int n, int key) {

    int s = 0, e = n-1;
    int mid = s + (e-s)/2;
    int ans = -1;
    while(s<=e) {

        if(arr[mid] == key){
            ans = mid;
            s = mid + 1;
        }
        else if(key > arr[mid]) {//Right me jao
            s = mid + 1;
        }
        else if(key < arr[mid]) {//left me jao
            e = mid - 1;
        }

        mid = s + (e-s)/2;
    }
    return ans;
}
```

Follow - up question :

Q. Count the number of occurances of an element in a sorted array.

→ (Last Occurance Index) - (Fist Occurance Index) + 1

Q. Peak in mountain array.

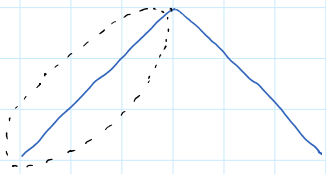
Eg: arr[] : {0, 10, 5, 2} → 

ans = 10.

Approach : If for 'mid' index,

1. arr[mid] < arr[mid+1] :

This element lies in the increasing slope / ascend. Search in the right half. ⇒ low = mid + 1

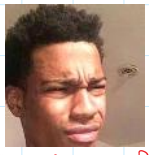


2. In all other cases.

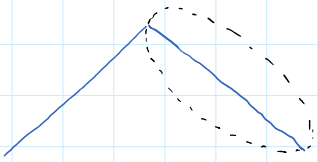
This element lies in the decreasing slope / descend. or is the peak element itself. Since mid can be the peak element, search in the left half including mid.

element, search in the left half including mid.

⇒ high = mid.



Confused?



Example: arr[] : {1, 10, 5, 2, 0}

① low = 0, high = 4, mid = 2

arr[] : {1, 10, 5, 2, 0}
l mid h

arr[mid] is not smaller than arr[mid+1]. ⇒ high = mid

② low = 0, high = 2, mid = 1

arr[] : {1, 10, 5, 2, 0}
l mid h

arr[mid] is not smaller than arr[mid+1]. ⇒ high = mid

③ low = 0, high = 1, mid = 0

arr[] : {1, 10, 5, 2, 0}
mid
l h

arr[mid] is smaller than arr[mid+1] ⇒ low = mid+1.

④ low = 1, high = 1, mid = 1

arr[] : {1, 10, 5, 2, 0}
mid
l h

Do this till **low < high**

Code:

```
int peakIndexInMountainArray(vector<int>& arr) {
    int s = 0;
    int e = arr.size() - 1;

    int mid = s + (e-s)/2;

    while(s<e) {
        if(arr[mid] < arr[mid+1]){
            s = mid + 1;
        }
        else
        {
            e = mid;
        }
        mid = s + (e-s)/2;
    }
    return s;
}
```

Homework: Find pivot in a rotated sorted array.

Discussed in the next Lecture. 😜

Selection Sort

24 February 2022 12:16

What is sorting ?

arr[] : {1, 7, 9, 2, 3, 0}

↓ sort

arr[] : {0, 1, 2, 3, 7, 9}

Arranging elements in a non-decreasing order is called sorting. Sorting can also be done in a non-increasing order.

What is Selection Sort ?

① You will make rounds/passes through the array.

② In each pass, you have to bring the smallest element to its correct place in the array.

③ You will then only consider the unsorted array excluding the smallest element you dealt with.

④ Repeat this until your array becomes sorted. (The array to sort will have only 1 element left).

Example: arr[] : {64, 25, 12, 22, 11}

1. {64 25 12 22 11}
   (0 1 2 3 4)
   i = 0.

   Search for the smallest element in this array and put it at i = 0.

2. {11 25 12 22 64}
   (0 1 2 3 4)
   i = 1

sorted Search for the smallest element in this array and put it at i = 1.

3. { 11 12 25 22 64 } (0 1 2 3 4)

Sorted Search for the smallest element in this array and put it at i = 2.

4. { 11 12 22 25 64 } (0 1 2 3 4)

Sorted Search for the smallest element in this array and put it at i = 3.

5. { 11 12 22 25 64 } (0 1 2 3 4)

Sorted 1 element left so it is definitely sorted.

Note: I made 4 passes for an array of length 5.

For each pass we will have to run a for loop from i = no. of elements put in their correct places to n-1 to get the smallest element from the unsorted array.

There will be n-1 passes

Code:

```
void selectionSort(vector<int>& arr, int n)
{
    for(int i = 0; i < n-1; i++ ) {
        int minIndex = i;

        for(int j = i+1; j<n; j++) {

            if(arr[j] < arr[minIndex])
                minIndex = j;

        }
        swap(arr[minIndex], arr[i]);
    }
}
```

Time Complexity:

$(n-1)$ passes:

Each pass has $(n-i-1)$ comparisons.
and 1 swap.

(so $(n-i)$ operations that are $O(1)$)

For $i=0$: n
$i=1$: $n-1$
$i=2$: $n-2$
⋮
$i=n-2$: 2

$\Rightarrow f(n) = n + (n-1) + (n-2) + \ldots + 2$

$\Rightarrow f(n) = 2 + 3 + \ldots + (n-1) + n.$

$\Rightarrow f(n) = \frac{n(n+1)}{2} - 1$

$\Rightarrow f(n) = \frac{n^2}{2} + \frac{n}{2} - 1$

$\Rightarrow f(n) = \boxed{O(n^2)}$

Space Complexity:

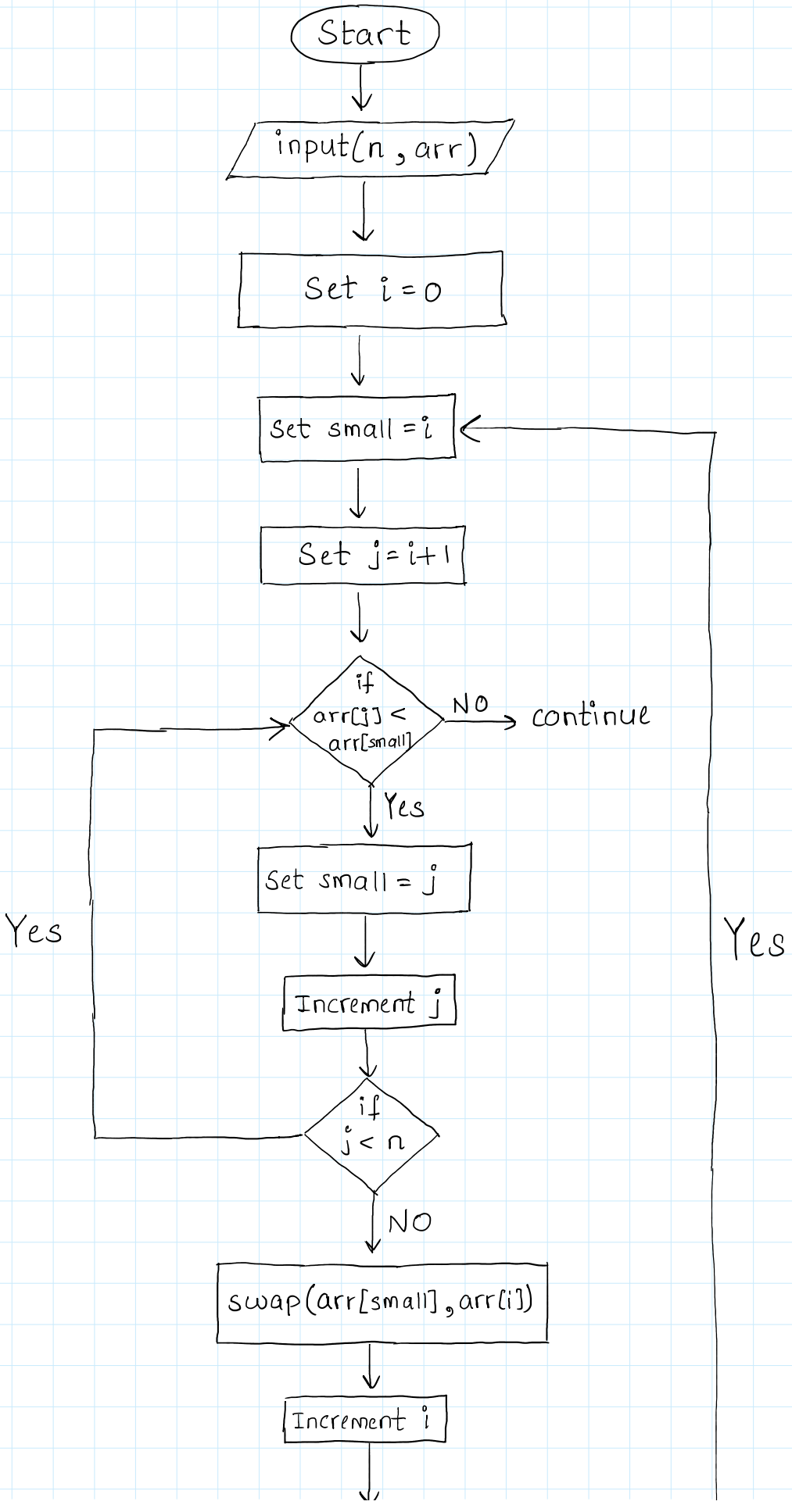
No extra memory/space has been used. Thus $\boxed{O(1)}$

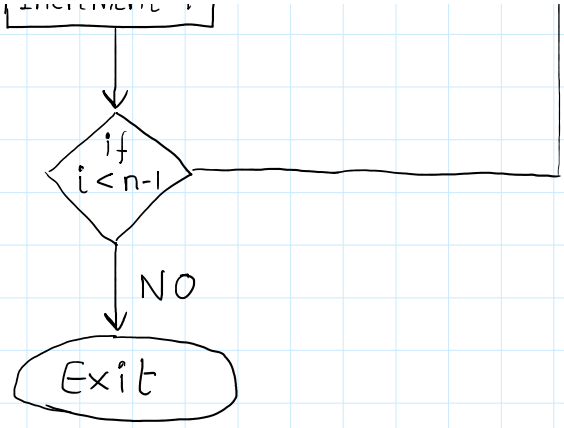
Use Cases:

① Works nicely with smaller arrays.

② When there are strict memory constraints.

Homework: Flow Chart for Selection Sort.





Homework: Is selection sort stable?

Ans: Since we make swaps after each pass in the unsorted array, we can have an array like

arr[] = {4, 2, 3, 1, 4}
(4 at index 0, 4 at index 1)

where the two 4's have an order which might get changed after sorting i.e. the (zero) 4 might appear after the (one) 4 in the sorted array.

Link: Is selection sort stable?

What is Recursion ?

→ When a function calls itself directly or indirectly, the process is called recursion and this call is called a recursive call.

Eg:
```
int fun(int n){
    fun(n);
}
```

If the solution of a problem depends on a smaller problem (subproblem) of the same type, then we will use recursion.

Eg: Find 2^n.

We can say that $2^n = \underbrace{2 \times 2 \times 2 \times \ldots \times 2}_{n \text{ times}}$

$\Rightarrow 2^n = 2 \times 2^{n-1}$

If we make a function that gives us 2^m when it's called like :

```
fun(m);
```

Then `fun(n) = 2 * fun(n-1);` ← Recurrence Relation

Eg: Find factorial (n!)

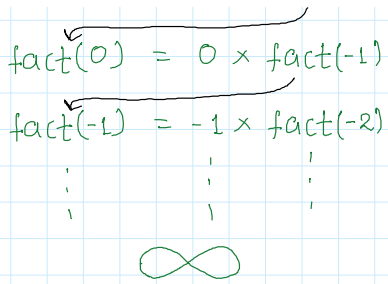
We know that $5! = 5 \times 4!$

$\Rightarrow fact(5) = 5 \times fact(4)$

$\Rightarrow fact(n) = n * fact(n-1)$

Given that we stop at a condition, example $0! = 1$, this is called base condition. Without this, our function will go bonkers !

$fact(3) = 3 \times fact(2)$
$fact(2) = 2 \times fact(1)$
$fact(1) = 1 \times fact(0)$



Step 1: Specify a base case where we return.

Step 2: Solve for the bigger problem using the smaller problem.

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

int factorial(int n) {

    //base case
    if(n == 0)
        return 1;

    return n * factorial(n-1);
}

int main() {

    int n;
    cin >> n;

    int ans = factorial(n);

    cout << ans << endl;

    return 0;
```

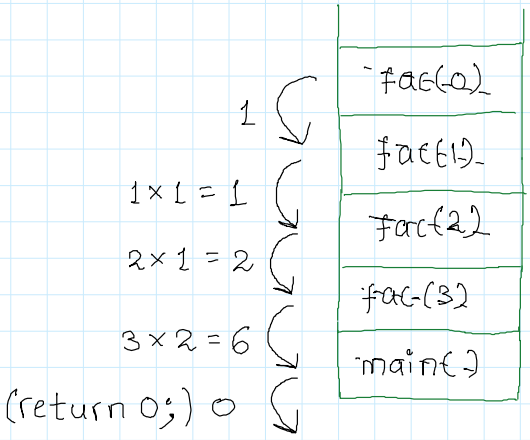
```
Users/lovebabbar/Documents/CodeHelp-DSA-Busted-Series/Lecture
31: Recursion Day1/" && g++ factroial.cpp -o factroial && "/U
sers/lovebabbar/Documents/CodeHelp-DSA-Busted-Series/Lecture3
1: Recursion Day1/"factroial
5
120
lovebabbar@192 Lecture31: Recursion Day1 % cd "/Users/lovebab
bar/Documents/CodeHelp-DSA-Busted-Series/Lecture31: Recursion
 Day1/" && g++ factroial.cpp -o factroial && "/Users/lovebabb
ar/Documents/CodeHelp-DSA-Busted-Series/Lecture31: Recursion
Day1/"factroial
4
24
lovebabbar@192 Lecture31: Recursion Day1 % cd "/Users/lovebab
bar/Documents/CodeHelp-DSA-Busted-Series/Lecture31: Recursion
 Day1/" && g++ factroial.cpp -o factroial && "/Users/lovebabb
ar/Documents/CodeHelp-DSA-Busted-Series/Lecture31: Recursion
Day1/"factroial
6
720
lovebabbar@192 Lecture31: Recursion Day1 %
```

Without a base case, we will get 'Segmentation Fault' which means you have exhausted your function call stack (the memory where function calls are accumulated when called).

```cpp
int factorial(int n) {

    //base case


    return n * factorial(n-1);
}
```

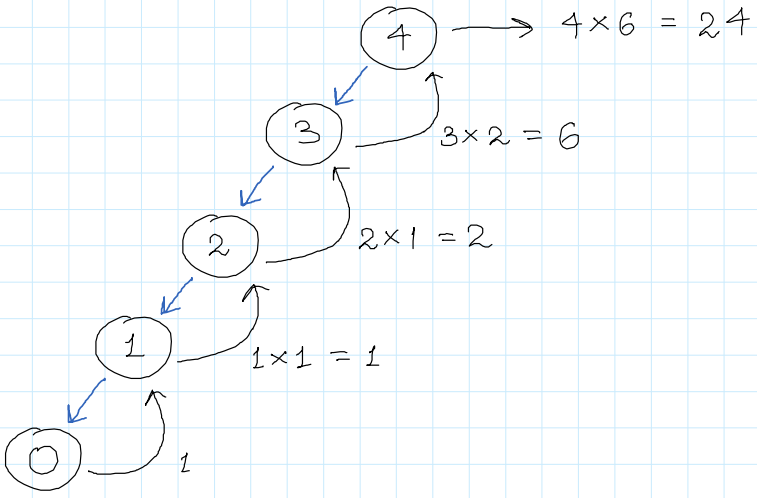
```
6
zsh: segmentation fault
lovebabbar@192 Lecture31: Recursion Day1 %
```

For fac(3), function call stack will look like this:



Recursion Tree :

Let's make a recursion tree for fac(4) where n = 4.



Final Structure of a Recursive Function :

```
Fun ( ) {
    Base Case
    Processing
    Recurrence Relation
}
```

Tail Recursion

```
Fun ( ) {
    Base Case
    Recurrence Relation
    Processing
}
```

Head Recursion

```cpp
int power(int n) {

    //base case
    if(n == 0)
        return 1;

    //recursive relation
    int smallerProblem = power(n-1);
    int biggerProblem = 2 * smallerProblem;

    return biggerProblem;
}
```

```
wer
10
1024
lovebabbar@192 Lecture31: Recursion Day1 % cd "/Users/lovebab
bar/Documents/CodeHelp-DSA-Busted-Series/Lecture31: Recursion
 Day1/" && g++ power.cpp -o power && "/Users/lovebabbar/Docum
ents/CodeHelp-DSA-Busted-Series/Lecture31: Recursion Day1/"po
wer
5
32
lovebabbar@192 Lecture31: Recursion Day1 %
```

OR

```cpp
int power(int n) {

    //base case
    if(n == 0)
        return 1;

    //recursive relation
    return 2 * power(n-1);
}
```

```
10
1024
lovebabbar@192 Lecture31: Recursion Day1 % cd "/Users/lovebabbar/Documents/CodeHelp-DSA-Busted-Series/Lecture31: Recursion Day1/" && g++ power.cpp -o power && "/Users/lovebabbar/Documents/CodeHelp-DSA-Busted-Series/Lecture31: Recursion Day1/"power
5
32
lovebabbar@192 Lecture31: Recursion Day1 % cd "/Users/lovebabbar/Documents/CodeHelp-DSA-Busted-Series/Lecture31: Recursion Day1/" && g++ power.cpp -o power && "/Users/lovebabbar/Documents/CodeHelp-DSA-Busted-Series/Lecture31: Recursion Day1/"power
6
64
lovebabbar@192 Lecture31: Recursion Day1 %
```

Example: Print n to 1.

Approach: Base case is when n = 0, else we will print the number and then call for n-1.

```cpp
void print(int n) {
    //base case
    if(n == 0) {
        return ;
    }

    cout << n << endl;

    print(n-1);
}
```

```
5
5
4
3
2
1
lovebabbar@192 Lecture30: Macros, Gloab Variable etc. %
```

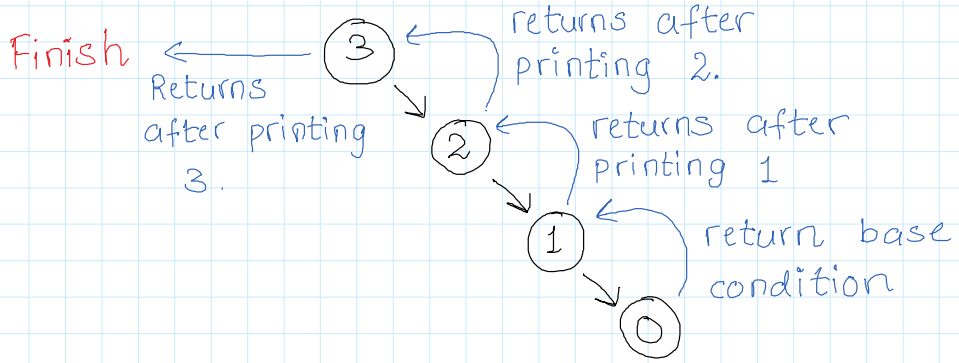
(Tail Recursion) → n to 1.

```cpp
void print(int n) {
    //base case
    if(n == 0) {
        return ;
    }

    //Recursive relation
    print(n-1);

    cout << n << endl;
}
```

```
5
1
2
3
4
5
lovebabbar@192 Lecture30: Macros, Gloab Variable etc. %
```

(Head Recursion) → 1 to n.

Recursion tree for 1 to n. (n = 3)



RECAP :

Recursive Function

- Mandatory:
  1. Base Case
  2. Recurrence Relation
- Optional:
  1. Processing

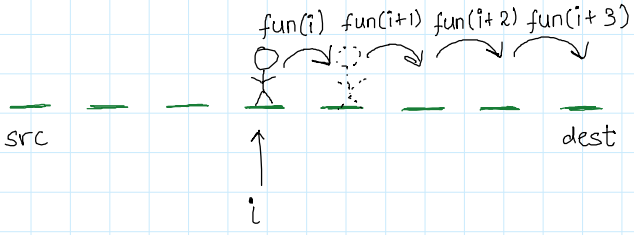
Solving a recursive problem involves 'leap of faith' wherein you solve one part / sub-problem and then expect your function to do the rest by calling it recursively.

Example : Go from source to destination.

src = 1 , dest = 10. (Use recursion)

Soln: Base Case is when src == dest.

At any given position, I will move one step ahead and then call myself again (Provided src != dest)




```cpp
void reachHome(int src, int dest) {

    cout << "source " << src  << " destination " << dest << endl;
    //base case
    if(src == dest) {
        cout << " pahuch gya " << endl;
        return ;
    }

    //processing - ek step aage badhjao
    src++;

    //recursive call
    reachHome(src, dest);

}
```

Example: Fibonacci Series

0, 1, 1, 2, 3, 5, 8, 13, ...

Print n^{th} term of Fibonacci Series.

Soln: $f(n) = f(n-1) + f(n-2)$

But, $f(n) = 0$ where $n <= 1$

and $f(2) = 1$

Baaki, $f(n) = f(n-1) + f(n-2)$ holds true.

```
class Solution {
public:
    int fib(int n) {
        //base case
        if(n == 0)
            return 0;

        if(n == 1)
            return 1;

        int ans = fib(n-1) + fib(n-2);

        return ans;

    }
};
```

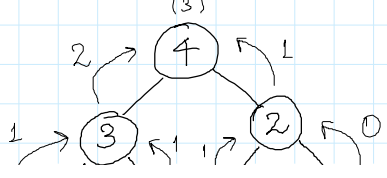
Homework: Solve using for loop.

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

int fib(int n) {
    if(n == 1 || n == 2) return n-1;
    int a = 0, b = 1;
    int ans;
    for(int i=3;i<=n;i++) {
        ans = a + b;
        a = b;
        b = ans;
    }
    return ans;
}

int main(void)
{
    int n;
    cin >> n;
    cout << "nth Fibonacci number is " << fib(n) << endl;
    return 0;
}
```

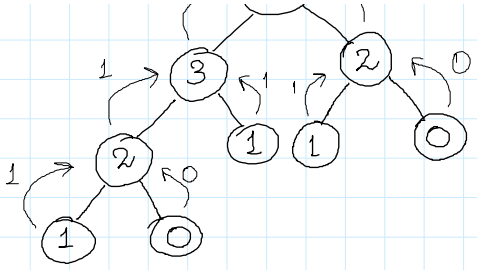
PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

```
10
nth Fibonacci number is 34
```

Recursion Tree for fibonacci :



(Zero-based index)



Example : Count ways to reach n^{th} stair.

Soln :

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

int climbStairs(int n) {
    if(n < 0) return 0;
    // only 1 way to reach the first stair
    // as you are already standing on it
    if(n == 0) return 1;

    // you came from the previous stair
    int penultimate = climbStairs(n-1);
    // you came from the stair before the previous stair
    int antepenultimate = climbStairs(n-2);
    return penultimate + antepenultimate;
}

int main(void)
{
    int n;
    cin >> n;
    cout << climbStairs(n) << endl;
    return 0;
}
```

Recursion tree is similar to Fibonacci.

Example : Say digits

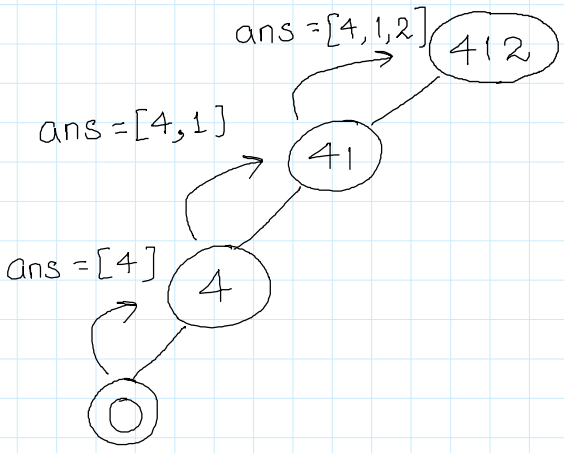
I/P - 412

O/P - "four" "one" "two"

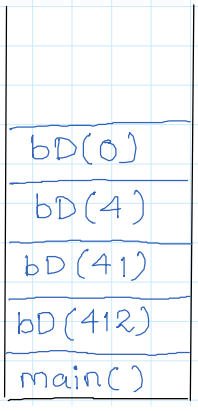
```cpp
#include <iostream>
#include <vector>
using namespace std;

void breakDigits(int n, vector<int>& ans) {
    if(n == 0) return;
    breakDigits(n/10, ans);
    ans.push_back(n%10);
}

void sayDigits(vector<int>& digits, string arr[]) {
    for(int i=0;i<digits.size();i++) {
        cout << arr[digits[i] - 1] << " ";
    }
    cout << endl;
}

int main(void)
{
    int n;
    cin >> n;
    vector<int> digits;
    breakDigits(n, digits);
    string arr[10] = {"one", "two", "three", "four", "five", "six", "seven", "eight",
    "nine"};
    sayDigits(digits, arr);
    return 0;
}
```

```
p\Work\Coding\Recursion\" ; if ($?) { g++ SayDigits.cpp -o SayDigits } ; if ($?) { .\SayDigits }
145236
one four five two three six
```

Recursion Tree : (Homework)



Function Call Stack :



Q1. Is Sorted : Check if array is sorted using Recursion.

Logic : Assume isSorted(v, i) checks if the array v[0....i] is sorted. So, at a given step, we have to check v[i] >= v[i-1] && isSorted(v, i-1) must be true.

Base Case - i == 0, one element is always sorted.

```
bool isSorted(vector<int>& v, int i) {
    if(v.size() == 0 || v.size() == 1)
        return true;
    if(i <= 0)
        return true;

    return v[i] >= v[i-1] && isSorted(v, i-1);
}
```

```
bool isSorted(int arr[], int size) {

    //base case
    if(size == 0 || size == 1 ){
        return true;
    }

    if(arr[0] > arr[1])
        return false;
    else {
        bool remainingPart = isSorted(arr + 1, size - 1 );
        return remainingPart;
    }

}
```

Babbar Code

Homework: Find sum of the elements of an array using Recursion.

Logic : Assume getSum(arr, n) gives the sum of the array arr containing n elements. Base case - n == 0 or n == 1.

```
int getSum(int arr[], int n) {
    if(n == 0) return 0;
    int sum = arr[0];
    sum += getSum(arr+1, n-1);
    return sum;
}
```

Call from main()

getSum(arr, n-1)

```
int sumOfArray(vector<int> v, int ind, int sum) {
    if(ind == v.size()) return 0;
    sum = v[ind];
    sum += sumOfArray(v, ind+1, sum);
    return sum;
}
```

Call from main()

sumOfArray(v, 0, 0)

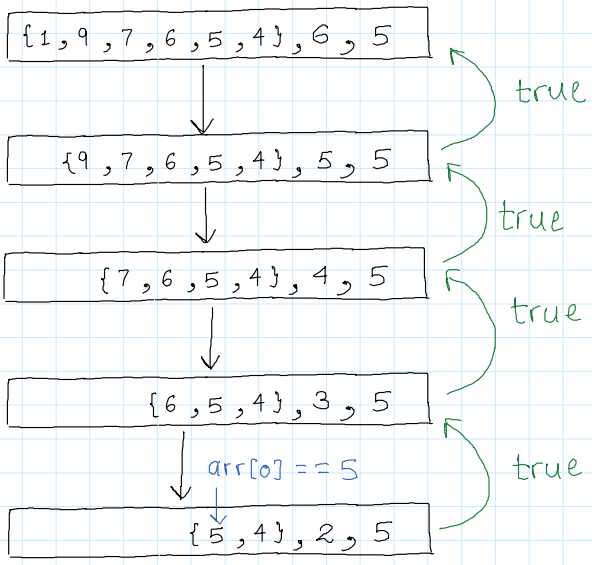
Q2. Search Element in array using Recursion. (Easy)

```
bool linearSearch(int arr[], int n, int val) {
    if(n < 0) return false;
    if(arr[0] == val)
        return true;
    // if the element is found even once, then we return true
    return linearSearch(arr+1, n-1, val);
}
```

Recursion Tree

Let arr[] = {1, 9, 7, 6, 5, 4}

val = 5 , n = 6



Binary Search Using Recursion :

```cpp
void printArr(int arr[], int l, int r) {
    cout << "Size of array is " << r-l+1 << endl;
    for(int i=l;i<=r;i++) {
        cout << arr[i] << " ";
    }
    cout << endl;
}

bool binarySearch(int arr[], int val, int l, int r) {
    printArr(arr, l, r);
    if(l>r) return false;

    int mid = l + (r-l)/2;

    if(arr[mid] == val)
        return true;
    else if(arr[mid] > val)
        return binarySearch(arr, val, l, mid-1);
    else if(arr[mid] < val)
        return binarySearch(arr, val, mid+1, r);

    return false;
}
```

```
10
1 3 4 5 7 8 11 12 13 14

14
```

Output.txt

```
Size of array is 10
1 3 4 5 7 8 11 12 13 14
Size of array is 5
8 11 12 13 14
Size of array is 2
13 14
Size of array is 1
14
Found
```

If you are not able to understand this, watch the video numbered 12 to understand Binary Search.

Homework : first and last Position using Recursion.

Iterative Approach

```cpp
// return the index of the first occurance of val in vector v
pair<int, int> firstAndLastPosition(vector<int>& v, int val) {
    int n = v.size();
    int l = 0, r = n-1, mid;
    pair<int, int> ans(-1, -1);
    while(l <= r) {
        mid = (l+r)/2;
        if(v[mid] == val) {
            ans.first = mid;
            r = mid - 1;
        }
        else if(v[mid] > val)
            r = mid - 1;

        else
            l = mid + 1;
    }
```

we can also break this down into 2 functions.

```cpp
// return the index of the first occurance of val in vector v
pair<int, int> firstAndLastPosition(vector<int>& v, int val) {
    int n = v.size();
    int l = 0, r = n-1, mid;
    pair<int, int> ans(-1, -1);
    while(l <= r) {
        mid = (l+r)/2;
        if(v[mid] == val) {
            ans.first = mid;
            r = mid - 1;
        }
        else if(v[mid] > val)
            r = mid - 1;

        else
            l = mid + 1;
    }
    if(ans.first == -1)
        return ans;

    l = ans.first, r = n-1;
    while(l <= r) {
        mid = (l+r)/2;
        if(v[mid] == val) {
            l = mid + 1;
            ans.second = mid;
        }

        else if(v[mid] > val)
            r = mid - 1;
    }
    return ans;
}
```

we can also break this down into 2 functions.

Recursive Approach

```cpp
int firstOcc(vector<int>& v, int l, int r, int val) {
    if(l > r) return -1;
    int n = v.size(), ans = -1;
    int mid = l + (r-l)/2;
    if(v[mid] == val) {
        ans = mid;
        int t = firstOcc(v, l, mid-1, val);
        if(t != -1)
            ans = t;
    }
    else if(v[mid] > val) {
        ans = firstOcc(v, l, mid-1, val);
    }
    else {
        ans = firstOcc(v, mid+1, r, val);
    }
    return ans;
}

int lastOcc(vector<int>& v, int l, int r, int val) {
    if(l > r) return -1;
    int n = v.size(), ans = -1;
    int mid = l + (r-l)/2;
    if(v[mid] == val) {
        ans = mid;
        int t = lastOcc(v, mid+1, r, val);
        if(t != -1)
            ans = t;
    }
    else if(v[mid] > val) {
        ans = lastOcc(v, l, mid-1, val);
    }
    else {
        ans = lastOcc(v, mid+1, r, val);
    }
    return ans;
}
```

Homework: Find total no. of occurances of an element in a sorted array.

Solution : (last occurance - first occurance + 1)

Homework : Find the peak in the mountain array.

Solution :

```
int findPeak(int arr[], int l, int r) {
    if(l > r) return -1;
    int ans = -1;
    int mid = (l+r) / 2;
    if(arr[mid] > arr[mid-1] && arr[mid] > arr[mid+1])
        return mid;

    else if(arr[mid] < arr[mid+1])
        return findPeak(arr, mid+1, r);

    return findPeak(arr, l, mid-1);
}
```

Homework : Find Pivot in a rotated sorted array.

Solution :

```
int findPivotRec(vector<int>& nums, int l, int r) {
    if(l > r) return 0;
    int mid = l + (r - l)/2;
    // given that we don't rotate the array k times which is a
    multiple of n.
    if(mid>0 && nums[mid] < nums[mid-1])
        return mid;

    else if(nums[mid] >= nums[0])
        return findPivotRec(nums, mid+1, r);

    return findPivotRec(nums, l, mid);
}
```

Homework : Search in rotated sorted array. (easy)

Solution : Find pivot and then use binary search in the half according to the value to be found.

Homework : Find square root using Binary Search.

```
long long int squareRoot(int n, int l, int r) {
    if(l > r) return 0;
    long long int mid = (l+r)/2;

    if(mid*mid <= n && (mid+1)*(mid+1)>n)
        return mid;

    else if(mid*mid > n) {
        return squareRoot(n, l, mid-1);
    }

    return squareRoot(n, mid+1, r);
}
```

RECURSION WITH STRINGS

Q1. Reverse a string using recursion.

I/P : "abcdc"
O/P : "cdcba"

I/P : "babbar"
O/P : "rabbab"

I/P : "abccba"
O/P : "abccba"

Logic : Solve 1 case and identify base cases.

Our function will look something like this :

```
void reverse (string s , int i , int j) {

}
```

where i & j are the 2 indices to be swapped.

Thus, base case will be when i > j.

We solve the given sub-task by swapping elements at i and at j, and then incrementing & decrementing i and j respectively. Call the function again till we hit base case.

Code :

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

void reverse(string& str, int i, int j ) {

    //base case
    if(i>j)
        return ;

    swap(str[i], str[j]);
    i++;
    j--;

    //Recursive call
    reverse(str,i,j);

}

int main() {

    string name = "babbar";

    reverse(name, 0 , name.length()-1 );

    cout << name << endl;

    return 0;
}
```

Don't forget to pass the string by reference or else we will not be modifying the original string name

Usecase: Let name = "abcde"

① reverse ("abcde", 0, 4)

↳ abcde ⇒ ebcda
i ⟷ j

② reverse("ebcda", 1, 3)

↳ ebcda ⇒ edcba
(i ⇄ j)

③ reverse("edcba", 2, 2)

↳ edcba ⇒ edcba
(i = j)

④ reverse("edcba", 3, 1)

↳ Base Case hits.

We can also make our base case hit for i >= j

Homework: Improve the code by using up just one extra variable i instead of 2 (i & j).

Logic: Same as above just use i to get the element to be swapped. We can use the string's size to get the element.

First Element : i

Second Element : n - i - 1.

Code :

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

void reverse(string& s, int i) {
    if(i >= s.size()/2)
        return;
    // swap the i and n-i-1 indexed nodes
    swap(s[i], s[s.size() - i - 1]);
    // call for the next index
    reverse(s, i+1);
}

int main(void)
{
    string s;
    cin >> s;
    reverse(s, 0);
    cout << s << endl;
    return 0;
}
```

```
PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE
ork\Coding\Recursion\" ; if ($?) { g++ ReverseString.cp
abcdefg
gfedcba
```

Q2 Check Palindrome. (If the reversed string is same as the original string)

I/P : "abccba"

O/P : true

I/P : "abcbde"

O/P : false

Approach 1: Reverse string and check if its equal to the original string.

Time Complexity : O(n)
Space Complexity : O(n)

Approach 2: Take 2 pointers i & j pointing to the first and last element. Check if they are equal & then increment and decrement i & j respectively till i < j. If at some point str[i] != str[j], return false.

Code :

```
bool checkPalindrome(string str, int i, int  j) {

    //base case
    if(i>j)
        return true;

    if(str[i] != str[j])
        return false;
    else{
        //Recursive call
        return checkPalindrome(str, i+1,j-1);
    }
}
```

Homework: Write the recursive function using just one extra pointing variable.

Code :

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

bool isPalindrome(string& s, int i) {
    if(i >= s.size()/2) return true;

    if(s[i] != s[s.size()-i-1])
        return false;

    return isPalindrome(s, i+1);
}

int main(void)
{
    string s;
    cin >> s;
    if(isPalindrome(s, 0)) {
        cout << "Palindrome\n";
    }
    else {
        cout << "Not Palindrome\n";
    }
    return 0;
}
```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

```
abccba
Palindrome
```

(Similar to reverse string)

Note: The time complexity of accessing the size of a std::string in C++ is O(1).

q3. Find power(a,b) using recursion.

I/P : 2, 4

O/P : 16

Approach : We can represent a^b as follows:

$$a^b = \begin{cases} a \cdot a^{b-1} & \text{if } b \text{ is odd} \\ a^{b/2} \cdot a^{b/2} & \text{if } b \text{ is even} \end{cases}$$

We can use a recursive function with base case being $b = 0$ (or $b = 1$), because $a^0 = 1$ (or $a^1 = a$)

For a given call, if $b \% 2 == 0$ then return func(a, b/2) * func(a, b/2),

else return a * func(a, b-1) which equals

a * func(a, b/2) * func(a, b/2).

Confused ?

Example : power(2,5)

$2^5 = 2 \times 2^4$

$2^4 = 2^2 \times 2^2$

$2^2 = 2^1 \times 2^1$

$2^1 = 2$

When power(2,5) is called, we will perform 2 steps in one call to improve efficiency.

Instead of giving back 2 * power(2,4)

we will calculate pow(2,2) before-hand by using variable = power(2, 5/2);

Then return 2 * variable * variable.

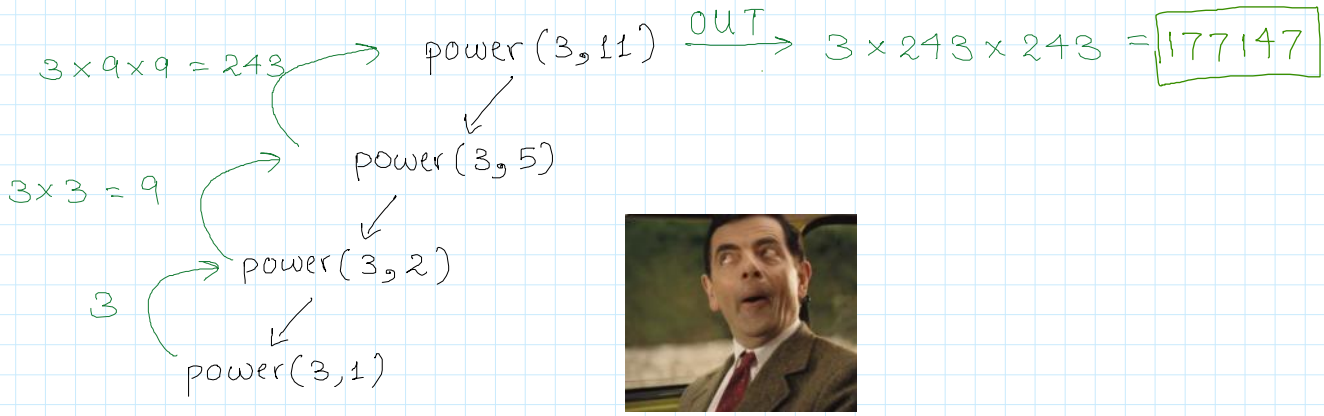
For even power,

return variable * variable.

Code :

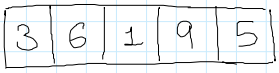
```
int power(int a, int b) {
    //base case
    if( b == 0 )
        return 1;

    if(b == 1)
        return a;

    //RECURSIVE CALL
    int ans = power(a, b/2);

    //if b is even
    if(b%2 == 0) {
        return ans * ans;
    }
    else {
        //if b is odd
        return a * ans * ans;
    }
}
```

Recursion Tree : power(3, 11)

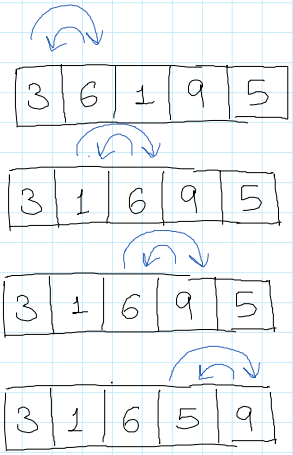


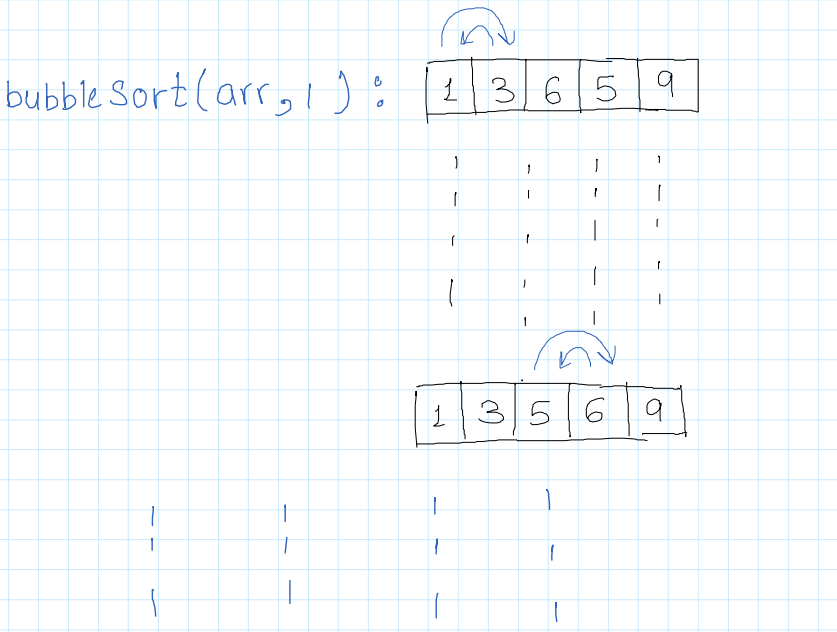
Q4 Bubble Sort using Recursion.

Approach : The recursive function will be called n times, each time it will correctly position the i^{th} largest element correctly. Base case hits when only 1 or no elements are left



bubbleSort(arr, 0) :





Code :

```
void sortArray(int *arr, int n) {

    //base case - already sorted
    if(n == 0 || n == 1) {
        return ;
    }

    //1 case sovle karlia - largest element ko end me rakh dega
    for(int i=0; i<n-1; i++) {
        if(arr[i] > arr[i+1]){
            swap(arr[i], arr[i+1]);
        }
    }

    //Recursive Call
    sortArray(arr, n-1);

}
```

Homework:

(1) Selection sort using recursion.

Code :

```
4   int minIndex(int a[], int i, int j)
5   {
6       if (i == j)
7           return i;
8       // minimum element's index from a[(i+1)....j]
9       int k = minIndex(a, i + 1, j);
10      // Comparing with current element and updating answer
11      return (a[i] < a[k])? i : k;
12  }
13
14  void recurSelectionSort(int a[], int n, int index = 0)
15  {
16      // Return when starting and size are same
17      if (index == n)
18          return;
19
20      // minimum element's index in a[index....(n-1)]
21      int k = minIndex(a, index, n-1);
22
23      swap(a[k], a[index]);
24
25      // Recursively calling selection sort function
26      recurSelectionSort(a, n, index + 1);
```

```
23      swap(a[k], a[index]);
24
25      // Recursively calling selection sort function
26      recurSelectionSort(a, n, index + 1);
27  }
28
```

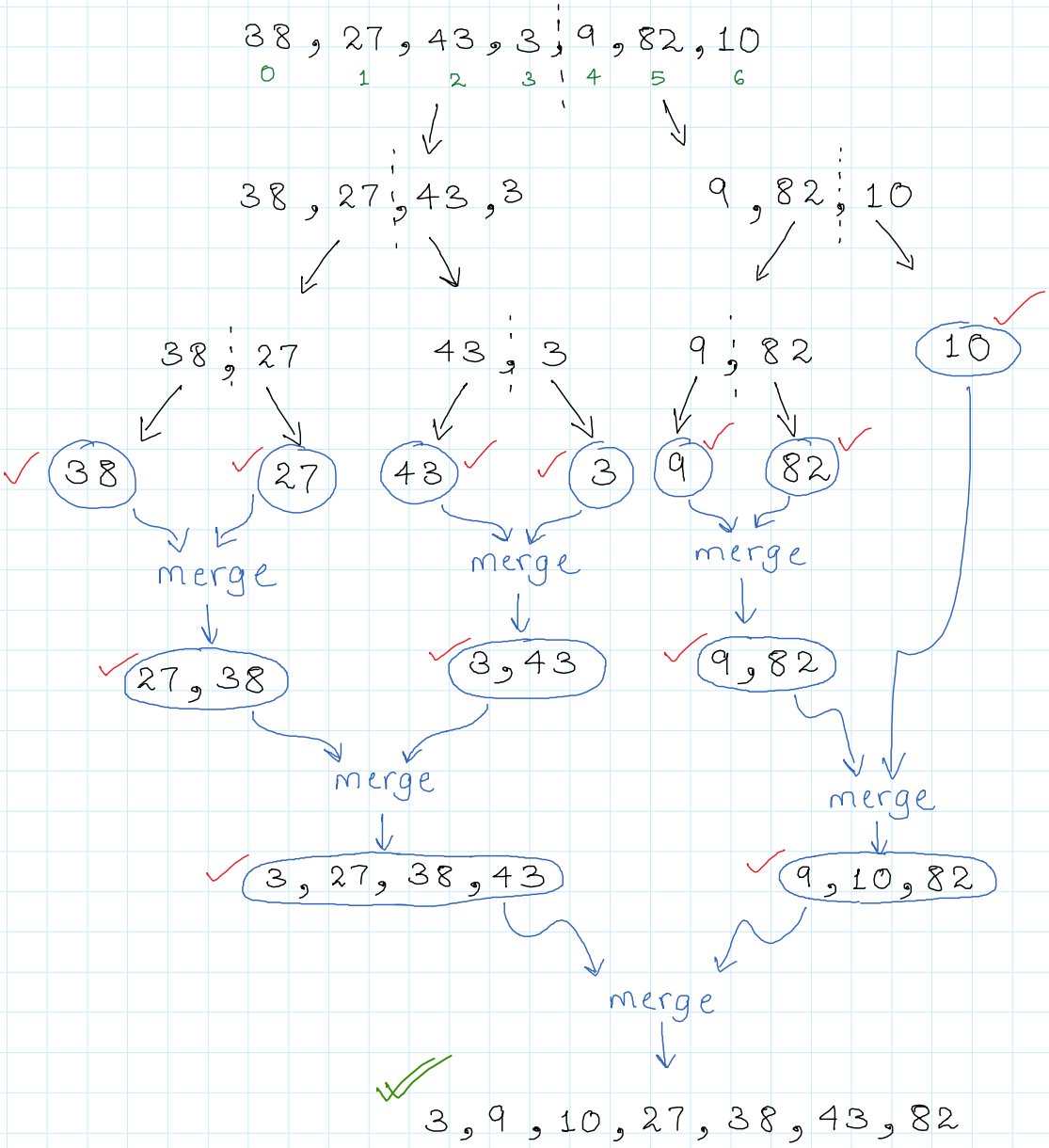
PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

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

Refer Here for detailed explaination of Homework Questions.

Merge Sort is a sorting algorithm that uses the concept of divide and conquer.

Eg:



✓ - Sorted Array.

Visualization (Source: Hacker Earth)

Approach 1 :

Logic - We will divide / split the array into 2 parts, call the function for the 2 parts recursively (with the

hope that this will return two sorted sub-parts).

We then merge the 2 sorted arrays and return the new sorted array.

Our base case is when we have 1 element in the array to be sorted, i.e. start >= end, we should return without doing anything.

Process : arr : 38 , 27 , 43 , 3 , 9 , 82 , 10
0 1 2 3 4 5 6

mergeSort(arr, 0, 6) is called for the entire array. This checks for the base case, (6-0+1) = length of the array = 7 ≠ 1, thus, base condition not satisfied.

We then call mergeSort(arr, 0, 3) & mergeSort(arr, 4, 6) with the faith that our 2 halves will get sorted.

We will then merge these 2 sorted arrays using merge(arr, 0, 6).

Code :

mergeSort function.

```
void mergeSort(int *arr, int s, int e) {

    //base case
    if(s >= e) {
        return;
    }

    int mid = (s+e)/2;

    //left part sort karna h
    mergeSort(arr, s, mid);

    //right part sort karna h
    mergeSort(arr, mid+1, e);

    //merge
    merge(arr, s, e);
}
```

helper merge function

```
void merge(int *arr, int s, int e) {

    int mid = (s+e)/2, len1 = mid - s + 1, len2 = e - mid;
    int *first = new int[len1];
    int *second = new int[len2];
    //copy values
    int mainArrayIndex = s;
    for(int i=0; i<len1; i++) {
        first[i] = arr[mainArrayIndex++];
    }

    mainArrayIndex = mid+1;
    for(int i=0; i<len2; i++) {
        second[i] = arr[mainArrayIndex++];
    }
    //merge 2 sorted arrays
    int index1 = 0;
    int index2 = 0;
    mainArrayIndex = s;
    while(index1 < len1 && index2 < len2) {
        if(first[index1] < second[index2]) {
            arr[mainArrayIndex++] = first[index1++];
        }
        else{
            arr[mainArrayIndex++] = second[index2++];
        }
    }
    while(index1 < len1) {
        arr[mainArrayIndex++] = first[index1++];
    }

    while(index2 < len2 ) {
```

```
        arr[mainArrayIndex++] = first[index1++];
    }

    while(index2 < len2 ) {
        arr[mainArrayIndex++] = second[index2++];
    }
    delete []first;
    delete []second;
}
```

Time Complexity : O(nlogn) → O(n) for every O(logn) partitions.

Space Complexity : O(n)

→ We are using 2 auxilliary arrays to merge the 2 sorted arrays.

Homework : Count inversions in an array.

Explaination - An inversion is when arr[i] > arr[j] for some i < j.

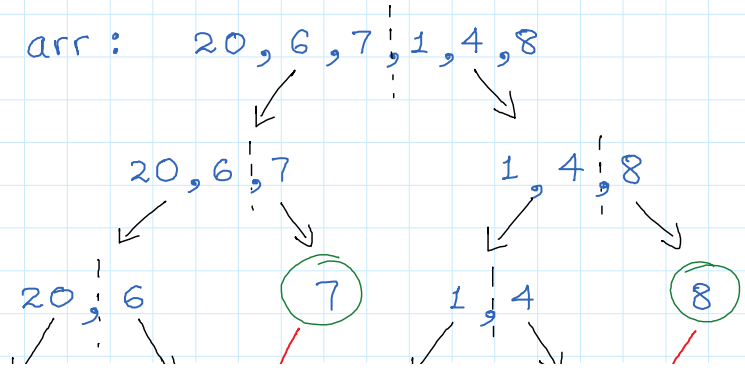
Example : 8 3 6 4 2

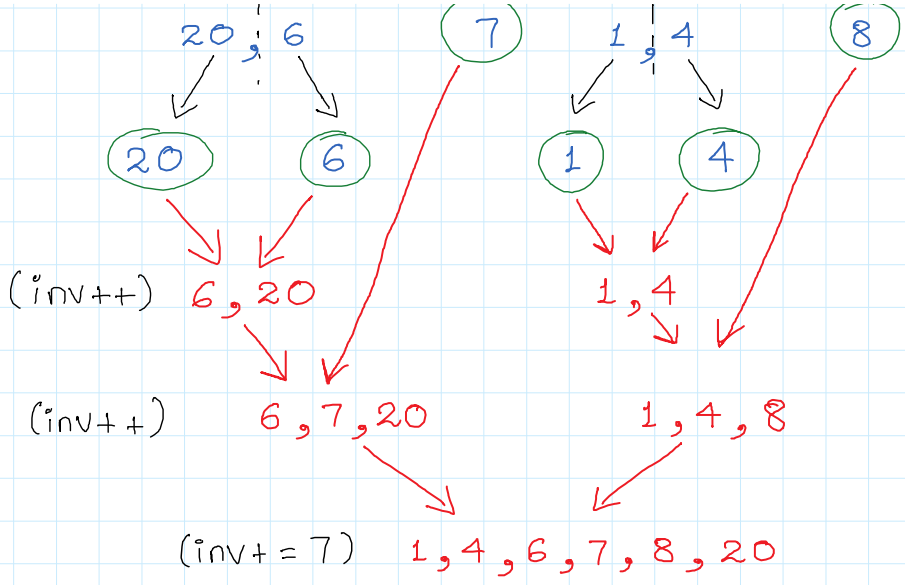
Inversions —
① 8 — 3
② 8 — 6
③ 8 — 4
④ 8 - 2
⑤ 3 — 2
⑥ 6 - 4
⑦ 6 — 2
⑧ 4 — 2

Answer = 8 (Elements occuring before, which are greater than element(s) after them.)

Logic : When we use the Merge Sort's partitioning approach, we merge the sorted arrays. We use this merging to count the number of inversions in the array.

Usecase : arr : 20, 6, 7, 1, 4, 8





answer = 9

For two sorted arrays | 6 | 7 | 20 | , | 1 | 4 | 8 |

i ↑ ↑ j ↑

① 6 > 1, so all elements in arr1 > 1. (arr1 is sorted)

⟹ inversion += 3 (length of arr1)

② 6 > 4, so again, all elements in arr1 > 4. (arr1 is sorted)

⟹ inversion += 3

③ 6 < 8, increment i. 7 < 8, increment i.

20 > 8, inversion += 1.

Logic is to fix j for all i and count the number of inversions. Repeat for all j.

Code:

inversionCount:

```
int inversionCount(vector<int>& nums, int l, int r) {
    if(l >= r) return 0;
    vector<int> temp(r - l + 1);
    int mid = l + (r - l)/2, inv = 0;

    inv += inversionCount(nums, 0, mid);
    inv += inversionCount(nums, mid+1, r);
    inv += merge(nums, l, mid, r, temp);

    return inv;

}
```

Counts the inversions for the split arrays and then for the merged arrays.

merge function:

counts the inversions while merging the 2 sub-arrays.

```cpp
int merge(vector<int>& nums, int l, int mid, int r, vector<int>& temp) {
    int i = l, j = mid+1, k = l;
    int inv = 0;
    while(i <= mid && j <= r) {
        if(nums[i] <= nums[j]) {
            temp[k++] = nums[i++];
        }
        else {
            temp[k++] = nums[j++];
            // all elements from nums[i] to nums[mid] will be greater than nums
            [j]
            inv += (mid - i + 1);
        }
    }
    while(i <= mid) {
        temp[k++] = nums[i++];
    }
    while(j <= r) {
        temp[k++] = nums[j++];
    }

    for(i=l;i<=r;i++) {
        nums[i] = temp[i];
    }
    return inv;
}
```

Intuition: Given an array arr[]: {3 6 1 4 2 5}, we choose an element (let's call it pivot) and place it at its correct position. We recursively call this function for the left and right parts where the elements are / maybe in wrong positions (The faith that our function will solve for the left and right sub-parts is the leap of faith).

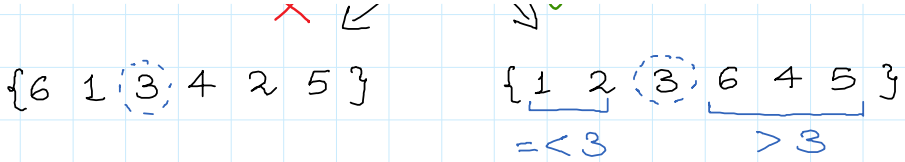
Example: arr[]: {3 6 1 4 2 5}
pivot

Let's say we keep the first element as our pivot. We must place 3 at its correct position. Find out how many elements are smaller than pivot. In our case, there are 2 elements 1 and 2 that are smaller than 3, so the correct position of 3 is index 2.

Note: Placing the pivot at its correct position also means that all elements to its left must be smaller and all elements to its right must be greater than the pivot (in no particular order).

arr[]: {3 6 1 4 2 5}

{6 1 3 4 2 5} {1 2 3 6 4 5}



$\Rightarrow$ arr[] = {1 2 3 6 4 5}

Call 1 Call 2

Code : Main code (Simple Recursive Calls)

1. Base Case : if low exceeds high.
2. Partition the array and return the correct index of the pivot element.
3. Repeat for the left half.
4. Repeat for the right half.

Partitioning Code

1. For a given array in the range arr[s.....e].
2. Take the pivot element as arr[s].
3. Count all element smaller than pivot.
4. Place/Swap pivot element with arr[s + count]
5. Start checking all elements with i=s & j=e till i < pivot & j > pivot. If arr[i] > pivot and arr[j] < pivot then swap them.

```cpp
void quickSort(int arr[], int s, int e) {
    if(s >= e) {
        return;
    }

    // The arr[partIndex] will now be at its correct position
    int partIndex = partition(arr, s, e);
    cout << "partIndex = " << partIndex << endl;

    // call the function for the left part
    quickSort(arr, s, partIndex-1);
    // call it for the right part
    quickSort(arr, partIndex+1, e);
}
```

```cpp
int partition(int arr[], int s, int e) {
    int pivot = arr[s];
    // count elements smaller than pivot
    int count = 0;
    for(int i=s+1;i<=e;i++) {
        if(arr[i] <= pivot)
            count++;
    }
    int pivotIndex = s + count;
    swap(arr[s], arr[pivotIndex]);
    // now arr[pivotIndex] is correctly placed
    int i = s, j = e;
    while(i < pivotIndex && j > pivotIndex) {
        if(arr[i] > arr[pivotIndex] && arr[j] <= arr[pivotIndex])
            swap(arr[i++], arr[j--]);
        else if(arr[i] < arr[pivotIndex]) {
            i++;
        }
        else {
            j--;
        }
    }
    return pivotIndex;
}
```

Homework :

① Is quick sort, an in-place sorting algorithm ?

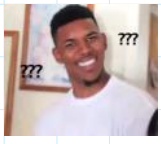
Ans: Yes. We are not creating secondary arrays and performing swaps in the original array.

② Is quick sort a stable sorting algorithm ?

Ans : Refer Here for detailed explanation

Time and Space Complexity

Time Complexity - Average Case : $O(n \log n)$
Worst Case : $O(n^2)$



Space Complexity - $O(n)$

Time Complexity of Recursive functions baadme padhaenge.

Hamari baat maan lo.

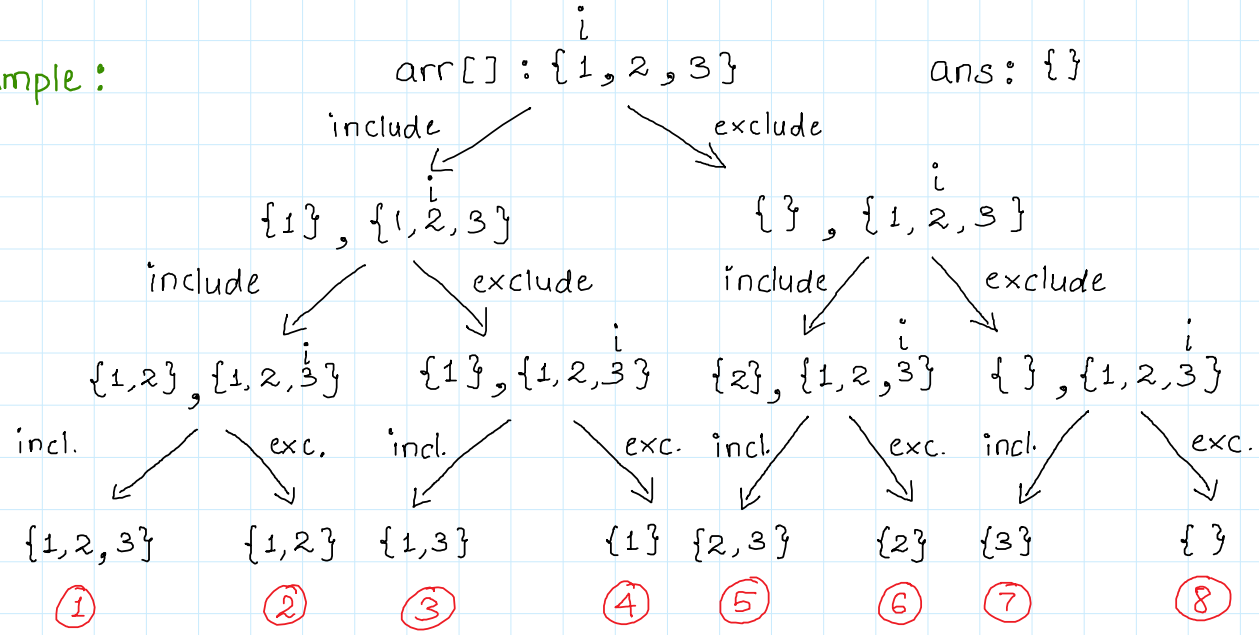
1. Generate all subsets (Power Set)

I/P : [1, 2, 3]
O/P : {{}, {1}, {2}, {3}, {1,2}, {2,3}, {1,3}, {1,2,3}}

Note - We can generate these subsets in any order as long as we cover all the subsets.

Approach : We can either include or exclude an element to generate a subset. Thus, we have 2 options for every element, either include it in the subset or exclude it.

Example :



What do you need ?

1. Array
2. Index
3. Answer array.

Code :

```
class Solution {
private:
    void solve(vector<int>& nums, vector<int> output, int i, vector<vector<int>> &ans) {
        if(i >= nums.size()) {
            ans.push_back(output);
            return;
        }
        // exclude this element and move on to the next index
        solve(nums, output, i+1, ans);
        // include this element
        output.push_back(nums[i]);
        solve(nums, output, i+1, ans);
    }
public:
    vector<vector<int>> subsets(vector<int>& nums) {
        vector<vector<int>> ans;
        vector<int> output;
        int ind = 0;
        solve(nums, output, ind, ans);
        return ans;
    }
};
```

Checking output array for each recursive call

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

void solve(vector<int>& nums, vector<int> output, int i, vector<vector<int>> &ans) {
    cout << "output: ";
    for(auto x: output) {
        cout << x << " ";
    }
    cout << endl;
    if (i == nums.size()) {
        ans.push_back(output);
        return;
    }
    // exclude this element and move on to the next index
    solve(nums, output, i + 1, ans);
    // include this element
    output.push_back(nums[i]);
    solve(nums, output, i + 1, ans);
}

vector<vector<int>> subsets(vector<int>& nums) {
    vector<vector<int>> ans;
    vector<int> output;
    int ind = 0;
    solve(nums, output, ind, ans);
    return ans;
}
```

```
3
1 2 3
```

Output.txt

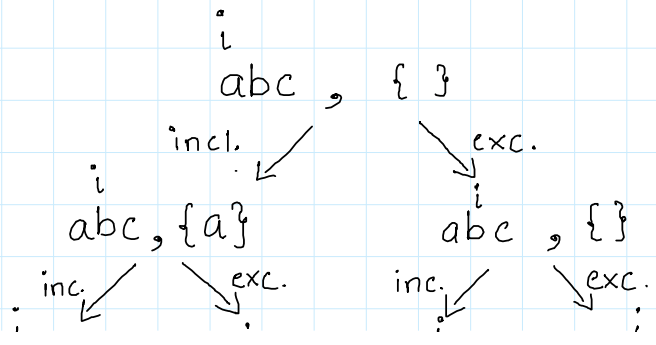
```
output:
output:
output:
output:
output: 3
output: 2
output: 2
output: 2 3
output: 1
output: 1
output: 1
output: 1 3
output: 1 2
output: 1 2
output: 1 2 3

3
2
2 3
1
1 3
1 2
1 2 3
```

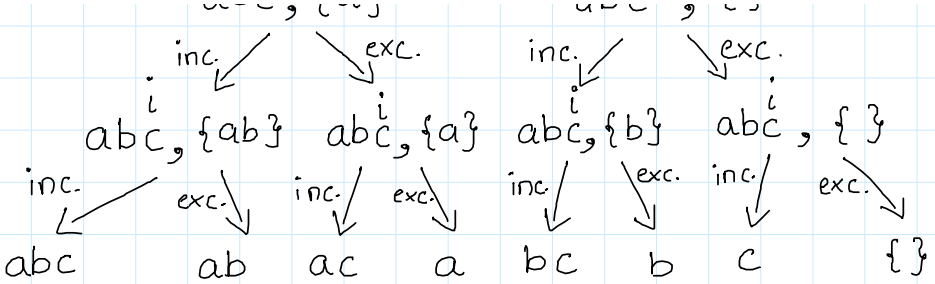
2. Subsequences of String. (Slight Change)

I/P : "abc"
O/P : {"", "a", "b", "c", "ab", "ac", "bc", "abc"}

Same Approach.

Recursion Tree :





Code :

```
void solve(vector<string>& ans, string str, string output, int i) {
    //base case
    if(i>=str.length()) {
        if(output.length()>0)
            ans.push_back(output);
        return;
    }

    //exclude
    solve(ans, str, output, i+1);
    //include
    output.push_back(str[i]);
    solve(ans, str, output, i+1);
}

vector<string> subsequences(string str){

    vector<string> ans;
    string output = "";
    solve(ans,str,output,0);
    return ans;

}
```

Note : In the question, we have to exclude the empty substring / subsequence.

Homework : Solve the above questions using bits.

Concept : If we have a string like "abc", then we know that 3-bit numbers 000, 001, ..., 111 represent all $2^3 = 8$ subsequences.

"abc" :

000	⇒	""
001	⇒	"c"
010	⇒	"b"
011	⇒	"bc"
100	⇒	"a"
101	⇒	"ac"

We 'mask' the binary number on the string.

Ex: "abc"
011

1 1 0 ⟹ "ab"

1 1 1 ⟹ "abc"

"bc"

So, for a string of length n, we will take an n-bit binary number and iterate through all the 2^n ranging from $\underbrace{000\ldots00}_{n}$ to $\underbrace{111\ldots11}_{n}$.

Code :

```
string maskedString(string str, int n) {
    string ans;
    int ind = str.length()-1;
    for(int j=0;j<str.length();j++) {
        if((1<<j) & n) {
            ans.push_back(str[ind]);
        }
        ind--;
    }
    return ans;
}

vector<string> subsequences(string str){

    vector<string> ans;
    for(int i=0;i<(1<<str.length());i++) {
        string temp = maskedString(str, i);
        if(!temp.empty())
            ans.push_back(temp);
    }
    return ans;
}
```

```
1 abc
```

Output.txt

```
1 c
2 b
3 cb
4 a
5 ca
6 ba
7 cba
8
```

Letter Combinations of a Phone Number

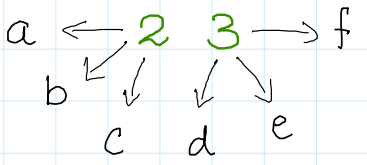
Pressing 2 can give "a", "b" or "c"

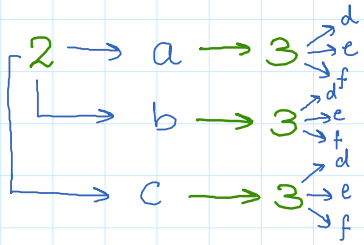
Pressing 23 can give : ad , ae , af , bd , be , bf , cd , ce , cf.

We have to return all possible combinations that can be formed by pressing a given phone number.

Note : Mapping exists only for numbers 2 - 9.

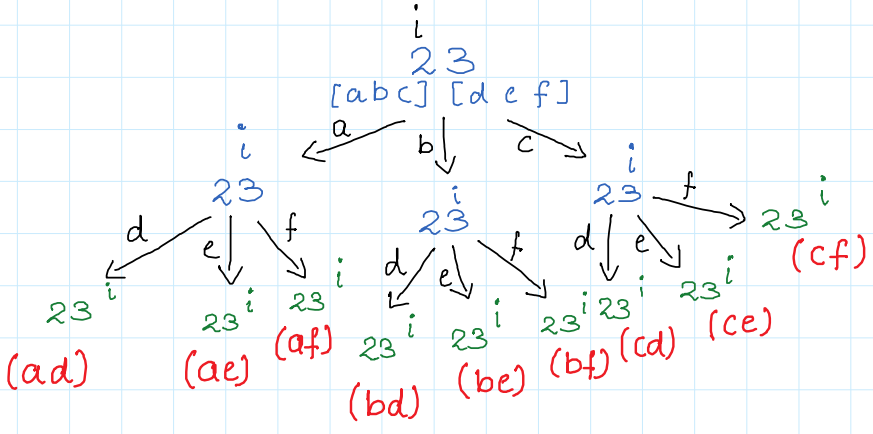
Approach : Just like subsets / subsequences (watch Lecture 37), we will iterate each digit in the phone number and for each alphabet belonging to the digit, we will choose it and move to the next digit recursively.





Example : Phone Number - 23

[a,b,c], [d,e,f]

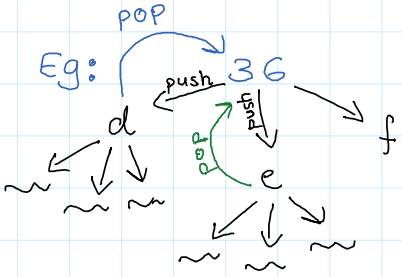


Code :

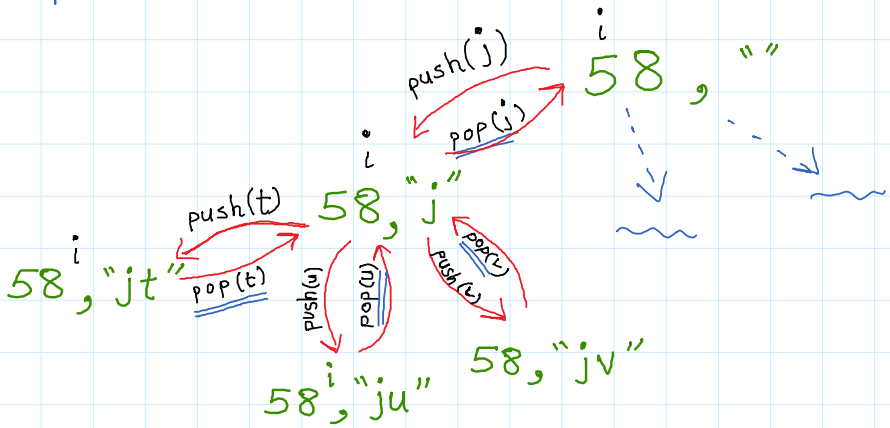
```
private:
    void solve(string digits, string mapping[], vector<string> &ans, string output, int i) {
        if(i >= digits.length()) {
            ans.push_back(output);
            return;
        }
        int number = digits[i] - '0';
        for(int j=0;j<mapping[number].length();j++) {
            output.push_back(mapping[number][j]);
            solve(digits, mapping, ans, output, i+1);
            output.pop_back();
        }

    }
public:
    vector<string> letterCombinations(string digits) {
        vector<string> ans;
        if(digits.length()==0)
            return ans;
        string output;
        int index = 0;
        string mapping[10] = {"", "", "abc", "def", "ghi", "jkl", "mno", "pqrs", "tuv", "wxyz"};
        solve(digits, mapping, ans, output, index);
        return ans;
    }
```

digits[i] will give me a character like "2" and not 2.

For all 'char' digits in the mapping of number, push it in our output string, get all combinations for this digit and then pop it to make way for the next digit.



Pop's Importance :



58 → [jkl, tuv]

PROBLEM STATEMENT:

We are given a string str, say "abc", we have to return all the permutations of the string in lexicographically increasing order.

A string str1 is lexicographically less than a string str2 if:

1. str1 ≠ str2
2. str1 is a prefix of str2

OR

2. There exists some i $(0 \le i \le \min(|str1|, |str2|))$ such that str1[i] < str2[i] (ASCII values) and for all $0 \le j < i$, str1[j] = str2[j].

Example: str1 = "abcdefg"
(indices 0 1 2 3 4 5 6)
str2 = "abcxyz"
(indices 0 1 2 3 4 5)

Equal — First Difference where str1[i] < str2[i]

⇒ str1 < str2 lexicographically

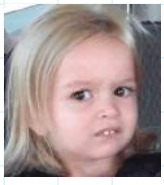
APPROACH: Say our string is "abc"

We want to generate the following permutations:

abc, acb, bac, bca, cab, cba

So, we want to swap elements throughout the string to generate the permutations.

Confused? No worries.



EXAMPLE: str = "abc"

Starting with i = 0, abc (i = 0, indices 0 1 2)

① Swap a with a, increment i. → a b c (i = 1)

② Swap a with b, increment i. → b a c (i = 1)

③ Swap a with c, increment i. → c b a (i = 1)

Call the function again for i = 1 for each output in steps ① to ③.

④ a b c (i = 1) ⇒ Swap b with b ⇒ a b c (i = 2) ①

⑤ a b c (i = 1) ⇒ Swap b with c ⇒ a c b (i = 2) ②

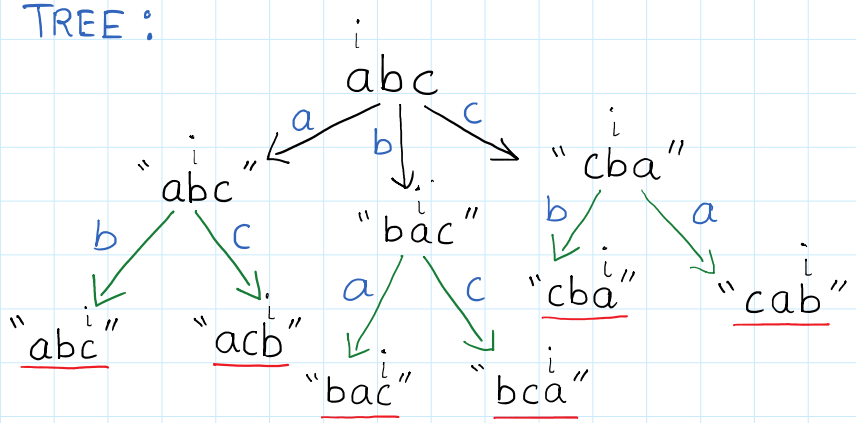
⑥ b a c (i = 1) ⇒ Swap a with a ⇒ b a c (i = 2) ③

⑦ b a c (i = 1) ⇒ Swap a with c ⇒ b c a (i = 2) ④

⑧ c b a (i = 1) ⇒ Swap b with b ⇒ c b a (i = 2) ⑤

⑨ c b a (i = 1) ⇒ Swap b with a ⇒ c a b (i = 2) ⑥

RECURSION TREE:

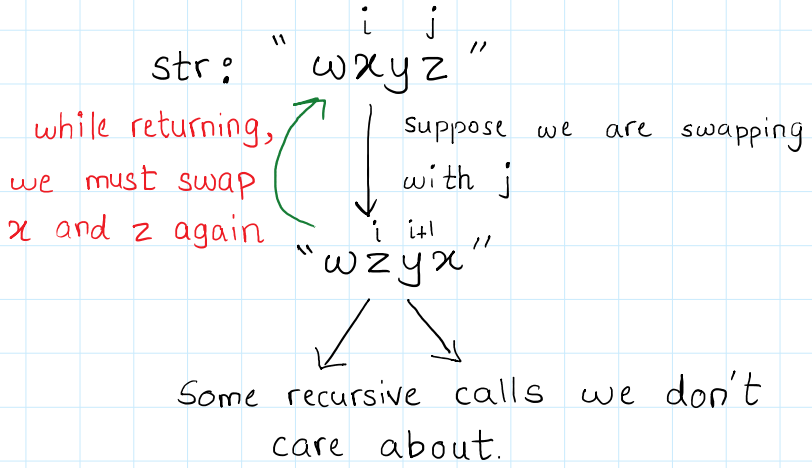


Code :

```
class Solution {
private:
    void solve(vector<int> nums, vector<vector<int>>& ans, int i) {
        if(i >= nums.size()) {
            ans.push_back(nums);
            return;
        }
        for(int j=i;j<nums.size();j++) {
            swap(nums[i], nums[j]);
            solve(nums, ans, i+1);
            swap(nums[i], nums[j]);
        }
    }
public:
    vector<vector<int>> permute(vector<int>& nums) {
        vector<vector<int>> ans;
        int index = 0;
        solve(nums, ans, index);
        return ans;
    }
};
```

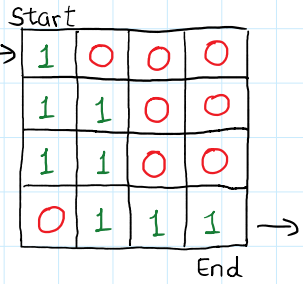
When you do a swap & generate all permutations for the given index, you need to swap them again to bring the array back to its original state.

EXAMPLE :



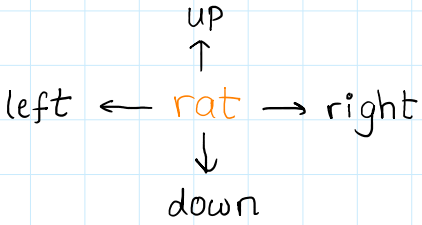
PROBLEM LINK : Click Here

Given :

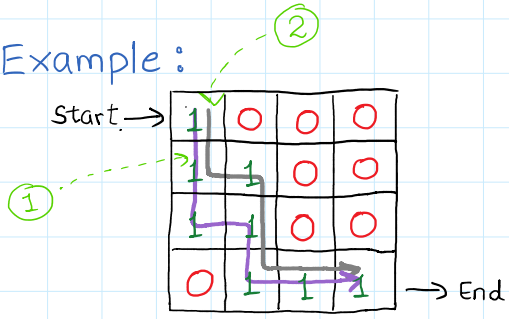


where : 1 - we can come to this cell
0 - we cannot come to this cell.

Aim is to output all possible ways to reach the End (n-1, n-1) from the Start (0, 0). The rat can move in all the 4 directions up, down, left, right.



Example :



1. DDRDRR
2. DRDDRR

APPROACH :

At any point, we have 4 choices – up, down, left or right.

For each position, check if you can go U/D/L/R keeping in mind the following constraints :

① Don't go out of bounds. ($0 \le i < n$, $0 \le j < n$)

② Don't go to an already visited position.

③ Don't go to a position marked 0.

③ Don't go to a position marked 0.

For point ②, maintain a 2-D array 'visited' which keeps track of all the visited positions. (nxn size)

Once you reach (n-1, n-1), you can print the string showing the path you took and then make all cells of the 'visited' matrix as False.

Thus, when you move, you will make visited[i][j] (if you moved to m[i][j]) and you will append your string with D, U, L or R according to your movement.

CODE: To check if a position is legit, we discussed some conditions above.

```
bool isSafe(int x, int y, int n, vector<vector<int>> visited, vector<vector<int>> &m ) {

    if( (x>=0 && x<n ) && (y>=0 && y<n) && visited[x][y] == 0 && m[x][y] == 1) {
        return true;
    }
    else {
        return false;
    }
}
```

In our main 'solve' function, we will first check if we have reached the end of the maze.

```
void solve(vector<vector<int>> &m, int n,vector<string>& ans, int x,
           int y, vector<vector<int>> visited , string path) {

    //you have reached x,y here

    //base case
    if( x == n-1 && y == n-1){
        ans.push_back(path);
        return;
    }
```

For the position (x, y), mark visited[i][j] = 1 (or true)

```
visited[x][y] = 1;
```

Now, for each direction (U, D, L, R) make newx & newy and make a move if going in that direction is safe.

if path is safe, then append the move in the path string.

```
if(isSafe(newx, newy, n, visited, m)) {
    path.push_back('D');
    solve(m,n,ans,newx,newy,visited,path);
    path.pop_back();
}
```

Call for the new position

After the recursion call returns, remove this move to make way for the next option. (U/D/L/R)

This gets called 4 times:

1. newx = x, newy = y-1
2. newx = x-1, newy = y
3. newx = x, newy = y+1
4. newx = x+1, newy = y.

```
int newx = x+1;
int newy = y;
if(isSafe(newx, newy, n, visited, m)) {
    path.push_back('D');
    solve(m,n,ans,newx,newy,visited,path);
    path.pop_back();
}

//left
newx = x;
newy = y-1;
if(isSafe(newx, newy, n, visited, m)) {
    path.push_back('L');
    solve(m,n,ans,newx,newy,visited,path);
    path.pop_back();
}

//Right
newx = x;
newy = y+1;
if(isSafe(newx, newy, n, visited, m)) {
    path.push_back('R');
    solve(m,n,ans,newx,newy,visited,path);
    path.pop_back();
}

//UP
newx = x-1;
newy = y;
if(isSafe(newx, newy, n, visited, m)) {
    path.push_back('U');
    solve(m,n,ans,newx,newy,visited,path);
    path.pop_back();
}
```

After this, we have covered all possible paths from (x,y) to (n-1, n-1). So, we will now backtrack and make visited[x][y] = 0 (or False)

```
visited[x][y] = 0;
```

What is time complexity ? (Discussed in Lecture 11)

→ Time complexity is the order of growth of a function measured against the input size.

✳ Recursive Function Examples :

① Factorial:

```
int factorial(int n) {
    if(n <= 1)
        return 1;
    return n * factorial(n-1);
}
```

Time Complexity = ??

Step 1 : Write the recurrance relation as a function.

$$f(n) = n * f(n-1)$$

Step 2 : Break down the function into components and include their respective Time complexities in a new function T(n), which is the total Time Complexity.

$$T(n) = \underbrace{\quad}_{\text{Part1}} + \underbrace{\quad}_{\text{Part 2}} + \text{----}$$

$$T(n) = K_1 + K_2 + T(n-1)$$

Part 1 : if {

```
int factorial(int n) {
    if(n <= 1)
        return 1;
    return n * factorial(n-1);
}
```

Part 3 : Recursive Call

Part 2 (Multiplication operation)

We will take all constant-time components together (k) and write the recurrance relation as a function of n.

$$T(n) = k + T(n-1) \quad \text{— (1)}$$

Repeat this for the smaller input(s) in eq (1)

$$T(n-1) = k + T(n-2)$$
$$T(n-2) = k + T(n-3)$$
$$\vdots$$
$$T(2) = k + T(1)$$
$$T(1) = 1 \quad \text{(Base Case)}$$

If you add up all these n equations, you get:

$$\begin{aligned} T(n) &= k + T(n-1) \\ T(n-1) &= k + T(n-2) \\ + \; T(n-2) &= k + T(n-3) \\ + \; T(n-3) &= k + T(n-4) \\ + \; T(n-4) &= k + T(n-5) \\ &\vdots \\ + \; T(2) &= k + T(1) \\ + \; T(1) &= 1 \end{aligned}$$

$$T(n) = \underbrace{k + k + \dots + k}_{n \text{ times}} + 1$$

$$T(n) = nk + 1 \approx nk \quad \text{(For large values of n)}$$

$$\boxed{T(n) = O(n)} \quad \text{(Order of n)}$$

② Binary Search:

Constant { Part 1 { Part 2 {

```
bool binarySearch(int arr[], int val, int l, int r) {
    if(l>r) return false;

    int mid = l + (r-l)/2;
```

Constant { Part 1 {
Part 2 {
Part 3 {
Part 4 {
Part 5 {
Constant { Part 6 {

```
    if(l>r) return false;

    int mid = l + (r-l)/2;

    if(arr[mid] == val)
        return true;
    else if(arr[mid] > val)
        return binarySearch(arr, val, l, mid-1);
    else if(arr[mid] < val)
        return binarySearch(arr, val, mid+1, r);

    return false;
}
```

Clearly,

$$f(n) = C + f(n/2)$$

(Only one of the 2 recursive functions will be called for each call)

$$T(n) = K + T(n/2)$$
$$+ \quad T(n/2) = K + T(n/4)$$
$$+ \quad T(n/4) = K + T(n/8)$$
$$+ \quad \vdots \quad \vdots \quad \vdots \quad \vdots$$
$$+ \quad T(2) = K + T(1)$$
$$+ \quad T(1) = 1$$

$$T(n) = a * K + 1$$

[where a is the no. of times we halved n to reach to the base case]

What is a ?

$$n \rightarrow n/2 \rightarrow n/4 \rightarrow n/8 \rightarrow \ldots \rightarrow 1$$

a times.

Using G.P:

first term = n

last term = 1

common ratio = $1/2$

$$a^{th} \text{ term} = n\left(\frac{1}{2}\right)^{a-1} = 1$$

$$\Rightarrow n = 2^{(a-1)}$$

$$\Rightarrow \log n = a - 1$$

$$\Rightarrow a \approx \log n \quad \text{(For large values of n)}$$

$$T(n) = (\log n) * k + 1$$

$$T(n) \approx k * \log n$$

$$T(n) = O(\log n)$$

③ Merge Sort :

Constant T.C. { Part 2 { Part 3 {

Part 4 {

```
void MergeSort(vector<int> &v, int l, int r)
{
    if(l >= r) return;
    int mid = l + (r - l)/2;
    MergeSort(v, l, mid);
    MergeSort(v, mid+1, r);

    merge(v, l, mid, r);
}
```

$$T(n) = k + T(n/2) + T(n/2) + \underbrace{nk'}_{\text{Part 4}}$$

Copying the merged array in the original array

$$T(n) = \underbrace{k}_{\text{ignore constant}} + nk' + 2T(n/2)$$

as it's insignificant as compared to nk'.

$$T(n) = nk' + 2T(n/2)$$

$$+ \left[T(n/2) = \frac{n}{2}k' + 2T(n/4)\right] \times 2$$

$$+ \left[T(n/4) = \frac{nk'}{4} + 2T(n/8)\right] \times 4$$

$$\vdots \qquad \vdots \qquad \vdots \qquad \vdots \qquad \vdots$$

$$+ \left[T(2) = 2k' + 2T(1)\right] \times \left(\frac{\log n}{2}\right)$$

$$+ \left[T(1) = 1\right] \times (\log n)$$

$$\Rightarrow \quad T(n) = nk' + 2T(n/2)$$

$$2 \times T(n/2) = nk' + 4T(n/4)$$

$$4 \times T(n/4) = nk' + 8T(n/8)$$

$$8 \times T(n/8) = nk' + 16T(n/16)$$

$$T(n) = \log_2 n \times nk' = \boxed{O(n \log n)}$$

④ Fibonacci Number :

$$f(n) = K + f(n-1) + f(n-2)$$

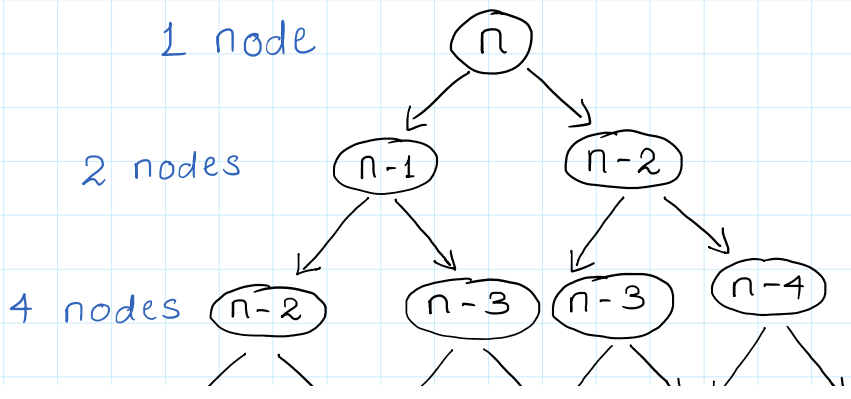
$$T(n) = K + T(n-1) + T(n-2)$$

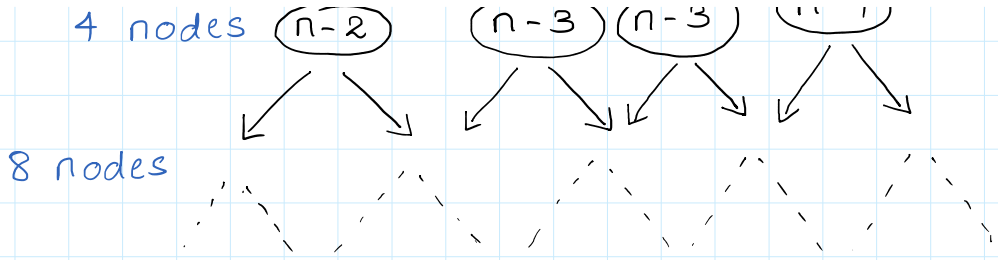
$$T(n-1) = K + T(n-2) + T(n-3)$$

$$T(n-2) = K + T(n-3) + T(n-4)$$

$$T(n-3) = K + T(n-4) + T(n-5)$$

Use a recursion tree for calculating Time Complexity.





2^n nodes.

Total number of nodes $= 1 + 2 + 4 + 8 + \ldots + 2^n$

$= 2^0 + 2^1 + 2^2 + 2^3 + \ldots + 2^n = 1\left(\frac{2^{n+1} - 1}{2 - 1}\right)$

$= (2^{n+1} - 1)$

If each node takes k amount of time,

$T.C = K \times 2 \times 2^n - K \approx 2K * 2^n \approx \boxed{O(2^n)}$

A few recursive questions :

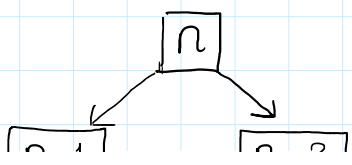
(1) Climb Stairs -

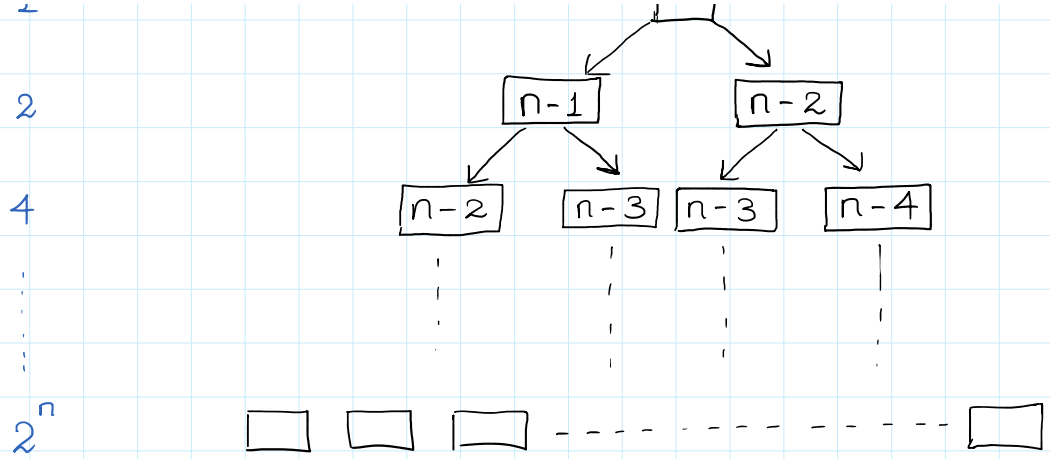
Const

```
int countDistinctWayToClimbStair(long long nStairs)
{
    //base case
    if(nStairs < 0)
        return 0;

    if(nStairs == 0)
        return 1;

    //R.C
    int ans = countDistinctWayToClimbStair(nStairs-1)
        + countDistinctWayToClimbStair(nStairs-2);

    return ans;
}
```

$T(n) = K + T(n-1) + T(n-2)$

1





Height of the tree = n.

For depth = i , no. of nodes = 2^i

Total no. of nodes = $1 + 2 + 4 + 8 + \ldots + 2^n$

$= 2^0 + 2^1 + 2^2 + 2^3 + \ldots + 2^n$

$= 1\left\{\frac{2^{n+1} - 1}{2 - 1}\right\} = (2^{n+1} - 1)$

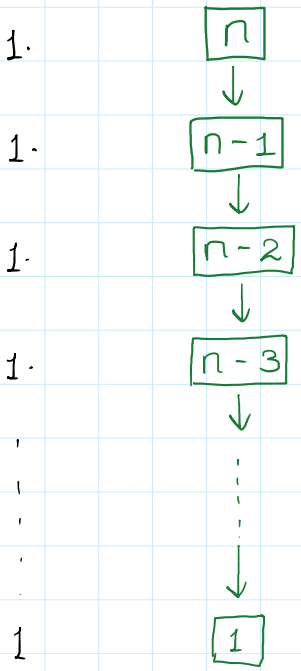
Time Complexity = $O(2^n)$ (Same as Fibonacci)

② Linear Search :

```
bool linearSearch(int arr[], int size, int k ) {

    //base case
    if(size == 0)
        return false;

    if(arr[0] == k) {
        return true;
    }
    else {
        bool remainingPart = linearSearch(arr+1, size-1, k );
        return remainingPart;
    }

}
```

Applies for `is array sorted' and other linear questions.

$T(n) = k + T(n-1)$



Total no. of nodes = $\underbrace{1+1+1+1+\ldots.+1}_{n \text{ times}}$

T.C. = n = $O(n)$

③ Finding Power :

Constant

Constant

```
int power(int a, int b) {
    //base case
    if( b == 0 )
        return 1;

    if(b == 1)
        return a;

    //RECURSIVE CALL
    int ans = power(a, b/2);

    //if b is even
    if(b%2 == 0) {
        return ans * ans;
    }
    else {
        //if b is odd
        return a * ans * ans;
    }
}
```

$T(n) = k + T(n/2)$ (Same as Binary Search)

T.C. = $O(\log n)$

④ Quick Sort :

```
void solve(vector<int>& arr, int s, int e) {

    //base case
    if(s >= e)
        return ;

    //partitioon karenfe
    int p = partition(arr, s, e);

    //left part sort karo
    solve(arr, s, p-1);

    //right wala part sort karo
    solve(arr, p+1, e);

}
```

In the best case, the pivot element will get placed in the middle after partitioning. This will become similar to merge sort T.C.

$$T.C. = nK + 2T(n/2)$$

$$\text{Time Complexity} = O(n \log n)$$

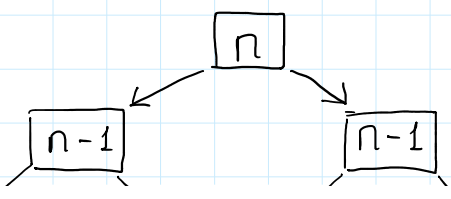
⑤ Subsequences / Subsets :

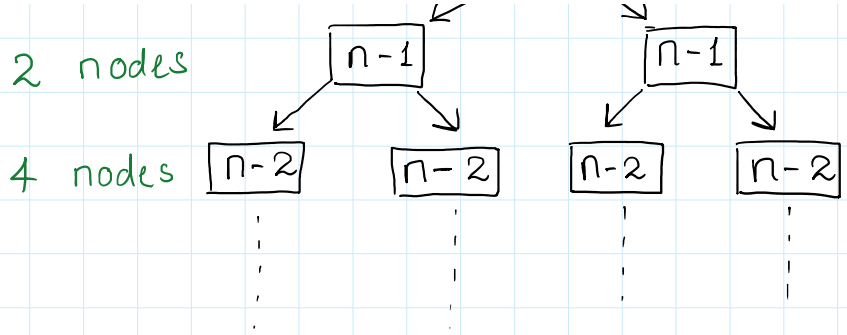
```
void solve(vector<int> nums, vector<int> output, int index, vector<vector<int> >& ans) {
    //base case
    if(index >= nums.size()) {
        ans.push_back(output);
        return ;
    }

    //exclude
    solve(nums, output, index+1, ans);

    //include
    int element = nums[index];
    output.push_back(element);
    solve(nums, output, index+1, ans);

}
```

$$T(n) = 2T(n-1) + K$$

1. node.

2 nodes





2^n nodes

Total nodes $= 1 + 2 + 4 + \ldots + 2^n$

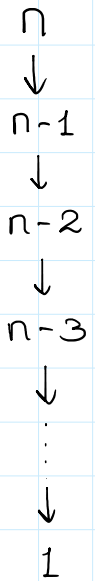
$= 1\left[\frac{2^{n+1} - 1}{2 - 1}\right] = (2^{n+1} - 1)$

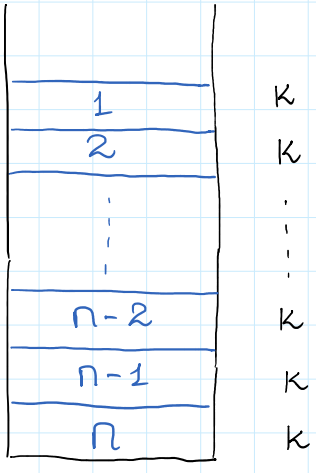
$T.C. = O(2^n)$

If you are printing / storing the elements of every subsequence, then $T.C. = O(2^n \times n)$

Space Complexity : Maximum space required at any instance.

(1) Factorial :





Call Stack Space

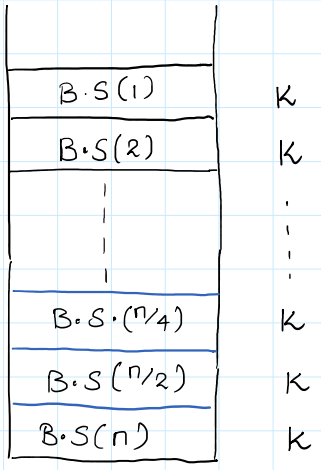
∵ Unless we reach base case $n=1$, all the calls will remain in the stack space, thus when we are calling for $n=1$, all n calls will be in the stack.

$$S.C. = k * n = O(n)$$

② Binary Search :

$S.C. = k * \log(n)$

$S.C = O(\log n)$



Call Stack Space

③ Merge Sort :

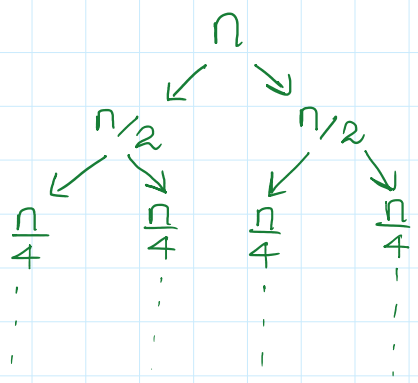
We are creating $2-\frac{n}{2}$ sized arrays for every n sized array that is used in a call.

Total = n

Total = n

Total = n

Total = n



$\log n$ height.

Space Complexity = $O(n \log n)$