MNNIT
COMPUTER CODING
CLUB

Class – 5

Revision and Recursion

Recap

OPERATORS

Arithmetic	+, -, *, /, %
Assignment	=
Increment/Decrement	++, --
Relational	<, >, <=, >=, ==, !=
Logical	&&, ||, !
Conditional	? :
Comma	,
Bitwise	&, |, ~, <<, >>, ^

OPERATOR	MEANING
++a	Increment a by 1,then use new value of a
a++	Use value of a ,then increment a by 1
--b	Decrement a by 1,then use new value of a
b--	Use value of a ,then decrement a by 1

CONTROL STATEMENTS

C++ provides two styles of flow control:

- Branching
- Looping

  Branching is deciding what actions to take and looping is deciding how many times to take a certain action.

IF ELSE STATEMENT

The `if` statement may have an optional `else` block. The syntax of the `if..else` statement is:

```
if (test expression) {
    // statements to be executed if the test expression is true
}
else {
    // statements to be executed if the test expression is false
}
```

IF ELSE LADDER

The if...else ladder allows you to check between multiple test expressions and execute different statements.

```
if (test expression1) {
   // statement(s)
}
else if(test expression2) {
   // statement(s)
}
else if (test expression3) {
   // statement(s)
}
.
.
else {
   // statement(s)
}
```

SW ITCH CASE

The switch statement allows us to execute one code block among many alternatives.

Syntax:

```
switch (expression)
{
     case constant1:
       // statements break;


   case constant2:
     // statements break;
   .
   .
   default:
     // default statements
}
```

Note:

The break statement is optional. If omitted, execution will continue on into the next case. The flow of control will fall through to subsequent cases until a break is reached.

LOOPS IN C++

a loop is used to repeat a block of code until the specified condition is met.

C++ programming has three types of loops:

- for loop
- while loop
- do...while loop

FOR LOOP

```cpp
#include<iostream>
using namespace std;
int main()
{
    for(int i=0;i<5;i++)
    {
        cout << "Value of i is: " << i << "\n";
    }
    return 0;
}
```

WHILE LOOP

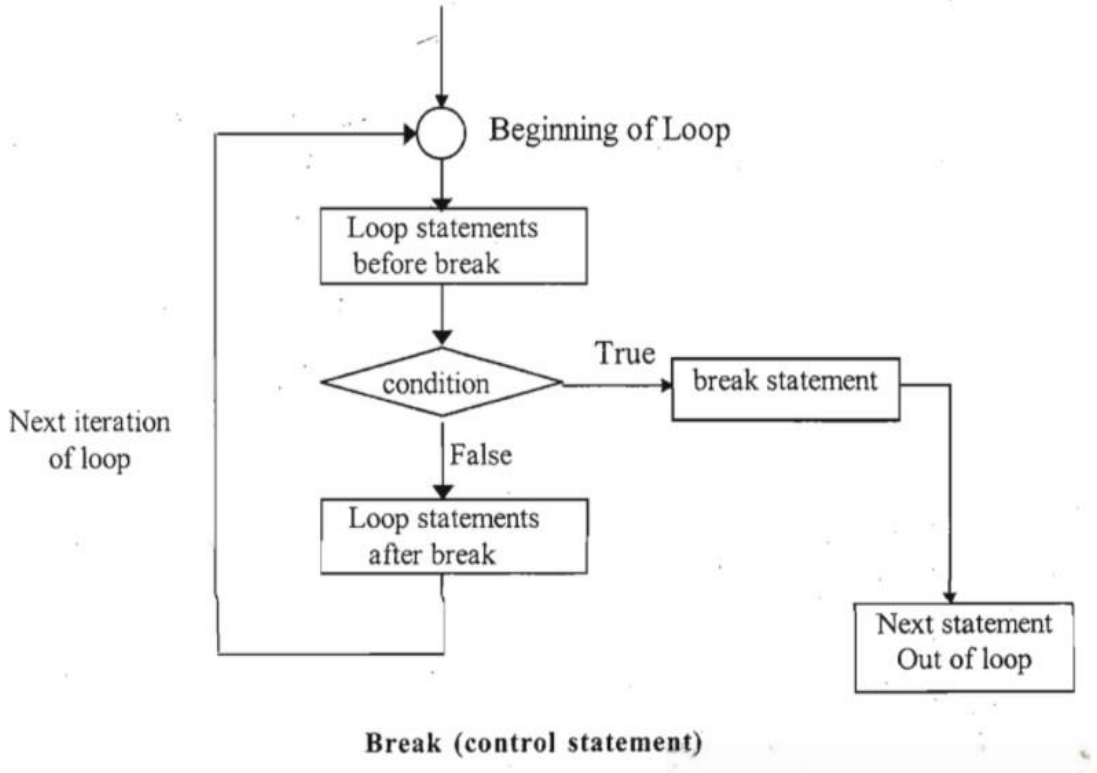
```cpp
#include<iostream>
using namespace std;
int main()
{
    int i = 0; //Initialisation
    while(i<5) //Test condition
    {
        cout << "Value of i is: " << i << "\n"; //Body
        i++; //Updation
    }
    return 0;
}
```

DO-WHILE LOOP

```cpp
#include<iostream>
using namespace std;
int main()
{
    int i = 0; //Initialisation
    do
    {
        cout << "Value of i is: " << i << "\n"; //Body
        i++; //Updation
    }
    while(i<5); //Test condition
    return 0;
}
```

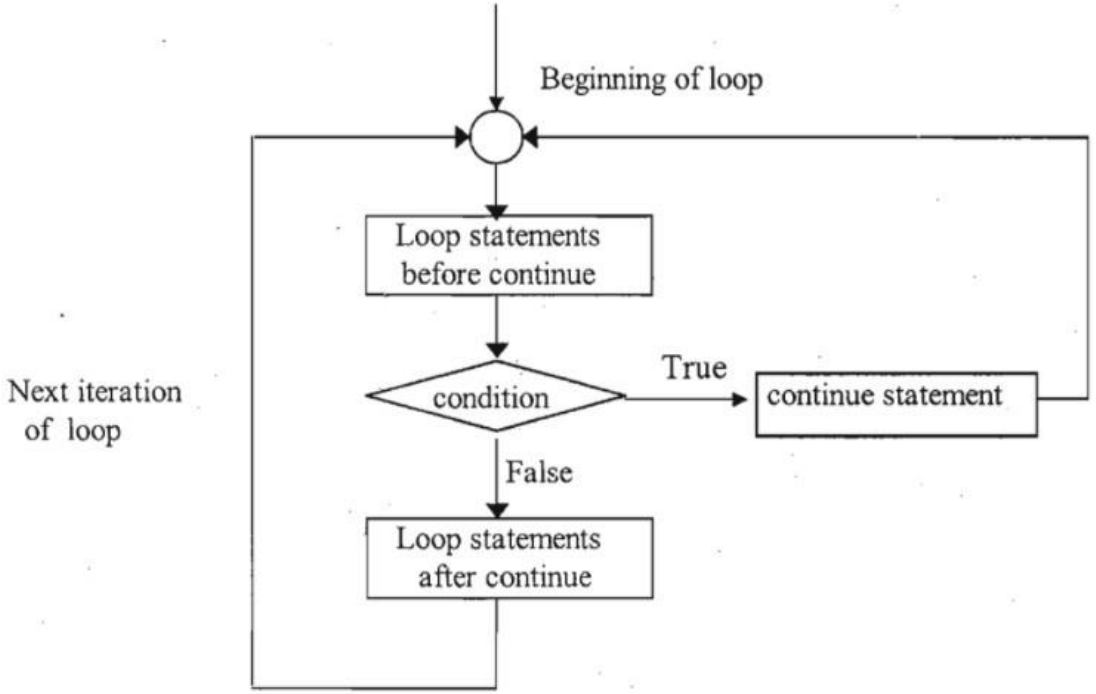
BREAK AND CONTINUE

Break: It is used to terminate the loop. This statement causes an immediate exit from that loop in which this statement appears, and the control is transferred to the statement immediately after the



Break (control statement)

Continue: It is used when we want to go to the next iteration of the loop after skipping some statements of the loop.



continue (control statement)

ARRAYS

•An array in C++ is a collection of homogenous items stored at contiguous memory locations.

•The elements of the array share the same variable name but each element has its own unique index number.
An array can be of any type. For example: `int`, `float`, `char` etc. If an array is of type `int` then it's elements must be of type `int` only.

```
int mark[5] = {19, 10, 8, 17, 9};
```

An array can also be initialized like

```
int mark[] = {19, 10, 8, 17, 9};
```

•Elements of array can be accessed by indices.

•Array index starts from 0 and goes till size of array minus 1.

2D ARRAY

An array of arrays is known as 2D array.

int x[3][4];

Here, 4 is the number of rows, and 3 is the number of columns.

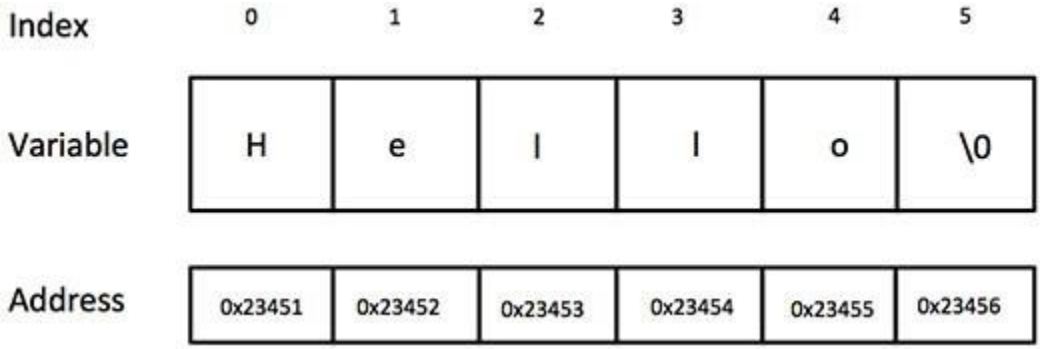
	Column 1	Column 2	Column 3	Column 4
Row 1	x[0][0]	x[0][1]	x[0][2]	x[0][3]
Row 2	x[1][0]	x[1][1]	x[1][2]	x[1][3]
Row 3	x[2][0]	x[2][1]	x[2][2]	x[2][3]

```
int arr[2][3] = {{ 1, 3, 0 }, { -1, 5, 9 }};
```

2D array can also be initialized like

```
int arr[2][3] = {1, 3, 0, -1, 5, 9};
```

CHAR ARRAY (STRINGS)



- Strings are actually one-dimensional array of characters terminated by a **null** character '\0'.

- `char greeting[6] = {'H', 'e', 'l', 'l', 'o', '\0'};`
-

```
char greeting[] = "Hello";
```

- Functions of Strings are found in string.h header file;

DEFINITION OF A FUNCTION

```
return_type function_name
        (parameters) { body of the
        function
}
```

Return Type − A function may return a value. The return_type is the data type of the value the function returns. Some functions perform the desired operations without returning a value. In this case, the return_type is the keyword void.

Function Name − This is the actual name of the function. The function name and the parameter list together constitute the function signature.

Parameters − A parameter is like a placeholder. When a function is invoked, you pass a value to the parameter. This value is referred to as actual parameter or argument.Parameters are optional; that is, a function may contain no parameters.

Function Body − The function body contains a collection of statements that define what the function does.

TYPES OF USER DEFINED FUNCTIONS

The functions can be classified into four categories on the basis of the arguments and return value

1. Functions with arguments and a return value
2. Functions with arguments and no return value
3. Functions with no arguments and a return value
4. Functions with no arguments and no return value

POINTERS

- The address of the first byte allocated to variable is known as address of variable.
- **'&' operator is the "address of" operator** in C++ which returns the address
- Eg -

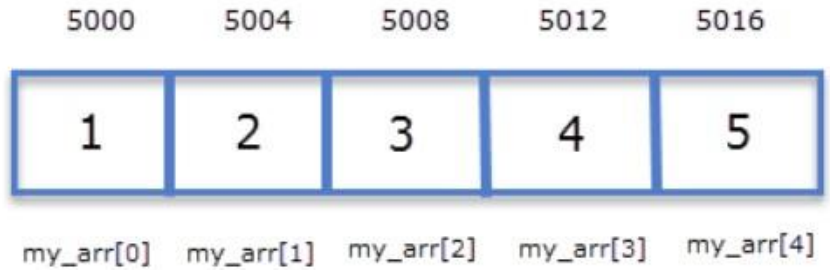
  int a = 5; - &a
  float f = 8.6; - &f

- The general syntax of declaration of pointer variable is: *datatype *p_name;*
- **'*' is used to dereference the pointer**
- Pointer variables are used to store the memory address. Used like normal variables.
- A pointer-to-pointer variable is used to store the address of a pointer variable

POINTER WITH 1D ARRAY

Consider an array

```
int arr[] = {1,2,3,4,5};
```



Here `arr` is a pointer to the first element aka arr is a pointer to int or (int*)

Remember
arr = &arr[0]
arr + 1 = &arr[1]
arr + 2 = &arr[2]
arr + 3 = &arr[3]

Thus
*(arr) = arr[0]
*(arr + 1) = arr[1]
*(arr + 2) = arr[2]
*(arr + 3) = arr[3]

FUNCTION CALLS

Call By Value: In this parameter passing method, values of actual parameters are copied to function's formal parameters and the two types of parameters are stored in different memory locations. So any changes made inside functions are not reflected in actual parameters of the caller.
Values of variables are passed by the Simple technique.

Call by Reference: Both the actual and formal parameters refer to the same locations, so any changes made inside the function are actually reflected in actual parameters of the caller.
Pointer variables are necessary to define to store the address values of variables.

RECURSION

- A technique where a function calls **itself** in loop.
- A powerful construct to solve complex problems easily
- Before writing a recursive function for a problem we should consider these points:
  - We should be able to define the solution of the problem in terms of a similar type of smaller problem.
  - At each step we get closer to the final solution of our original solution
  - There should be a terminating condition to stop recursion.

```
1   #include<stdio.h>
2
3   void func()
4   {
5       //Statement Before Calling
6       //Exit Condition
7       func();
8       //Statement before return
9   }
10
11  int main()
12  {
13      func();
14  }
```

PROBLEM: PRINT 1 TO N

Here are 2 variants of the problem.

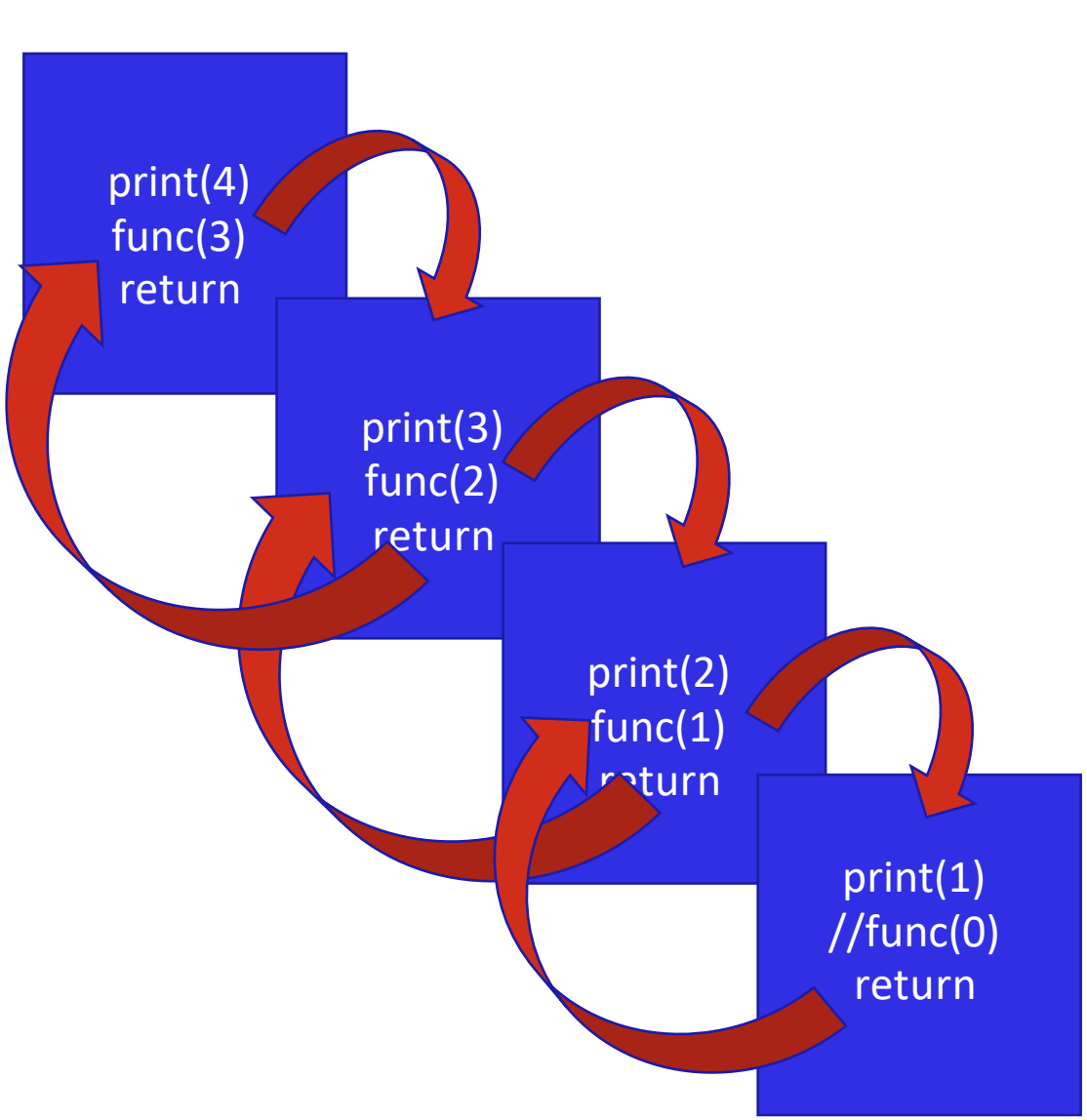
One where numbers are printed in ascending order and other with descending order.

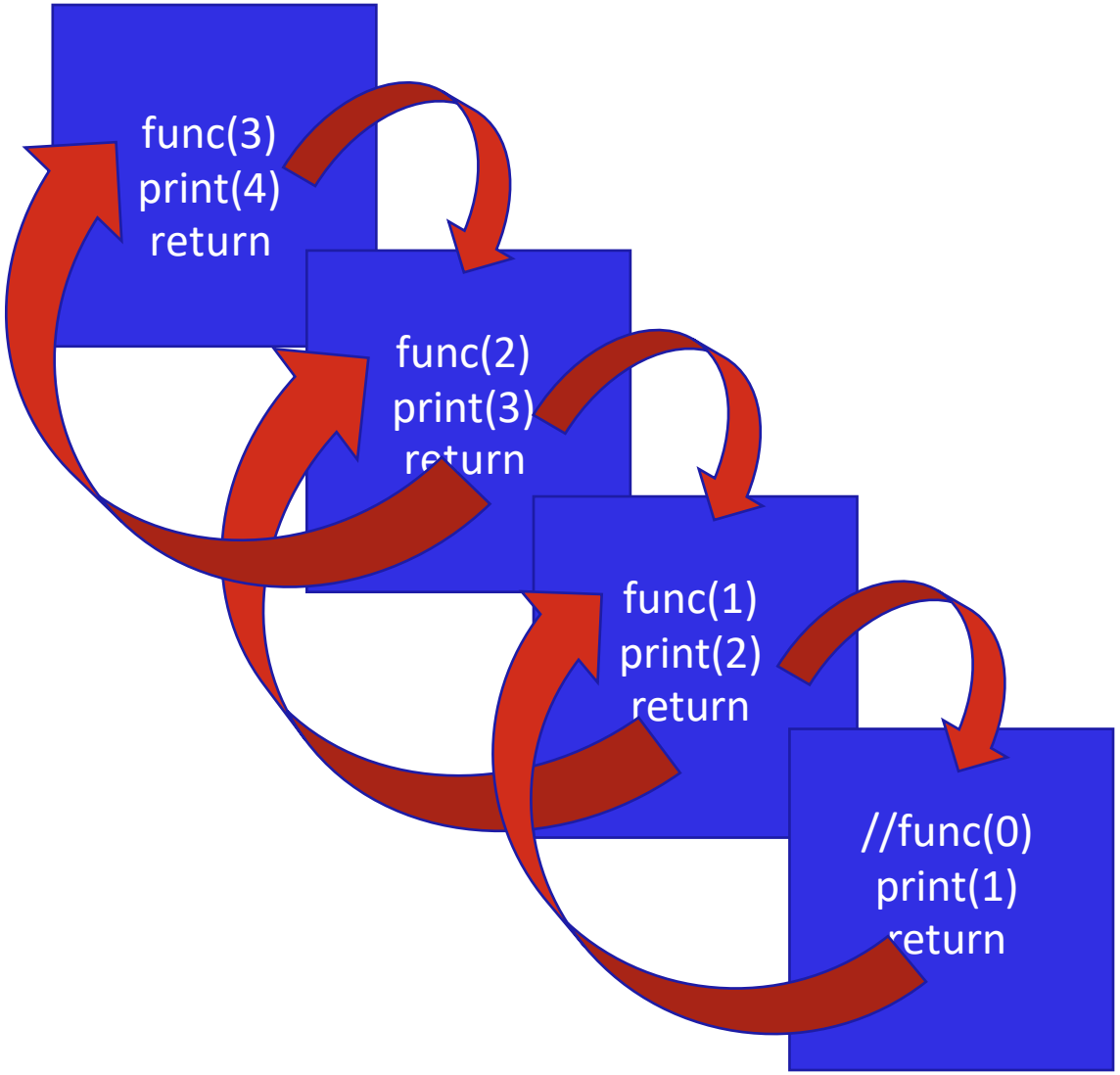
- Step 1: Determine the subproblem
- Step 2: Identify Exit Condition
- Step 3: Determine where to solve the current iteration (before recursive call or after recursive call)

```
void printNumbers(int n) {
    if(n>1)
        printNumbers(n-1);
    cout<<n<<" ";
}
```

```
void printNumbers(int n) {
    cout<<n<<" ";
    if(n>1)
        printNumbers(n-1);
}
```

HOW IT'S WORKING... ?



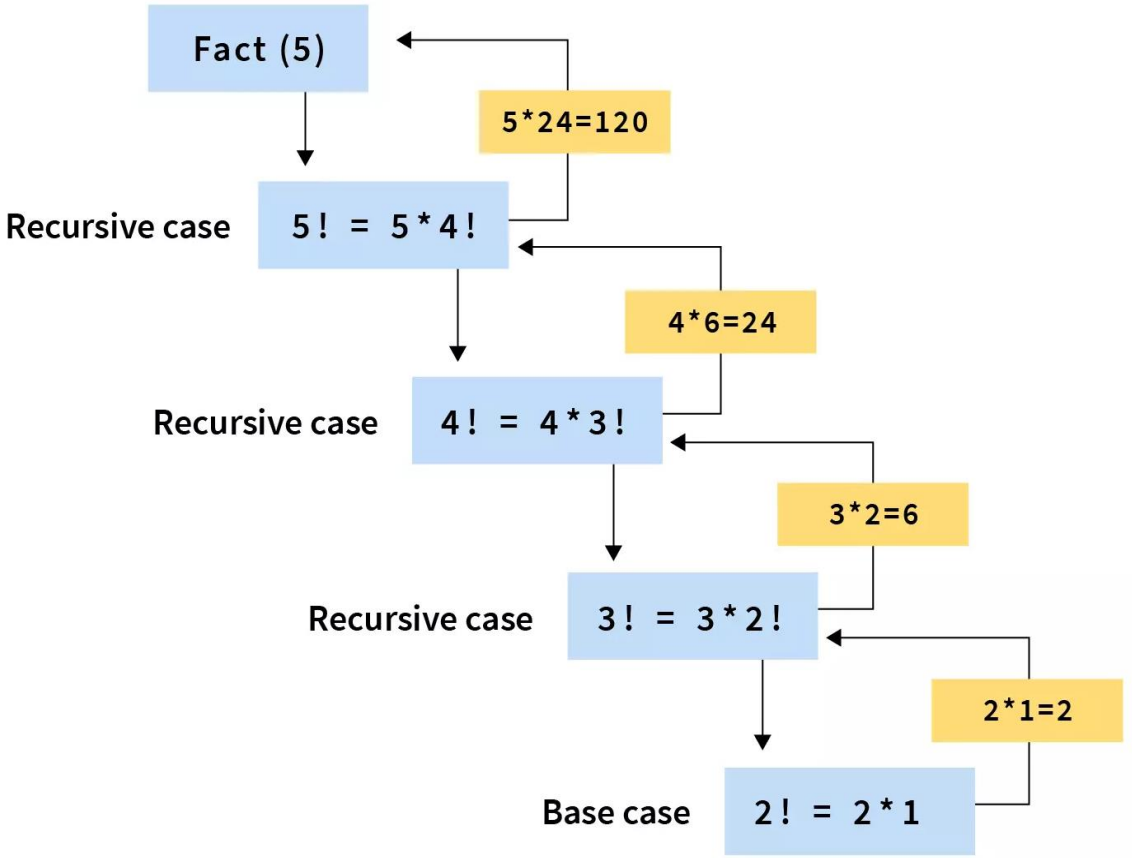


PROBLEM: FACTORIAL OF A NUMBER

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

int fact(int n) {
    if(n==1) return 1;
    return n*fact(n-1);
}
int main() {
    int n;
    cin>>n;

    cout<<fact(n);
    return 0;
}
```



Write code to print nth fibonacci number using recursion.

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

int fibonacci(int n){
  if(n<=1 ){
    return n;
  }
  return fibonacci(n-1)+fibonacci(n-2);
}
int main() {
  int n;
  cin>>n;
  cout<<fibonacci(n);
  return 0;
}
```

GCD

GCD stands for Greatest Common Divisor. It will the number which completely divide the given numbers and will be largest.

For example:
30 = **1, 2, 3, 5, 6, 15, 30**
Where 1, 2, 3, 5, 6, 15, 30 are divisors of 30.
6 = **1, 2, 3, 6**
Where 1, 2, 3, 6 are divisors of 6
Now from these two list of divisors we have to get greater number, which is **6.** So that GCD of 30, 6 will be **6.**

=> Write a program using loops to find GCD of two numbers.

Euclidian Algorithm

```
#include <iostream>
using namespace std;
int main() {
int num1, num2, temp;
cout << "Enter two numbers: ";
cin >> num1 >> num2;
while (num1%num2 != 0){
temp = num1;
num1 = num2;
num2 = temp%num2;
}
cout << “GCD is " << num2 << endl;
return 0;
}
```

C++ has inbuilt function to calculate GCD of two numbers.
This function uses Euclidian Algorithm to calculate GCD.

The function is:

__gcd(18,27)

Use header file
#include<algorithm> to use this function

<bits/stdc++.h> in C++

The <bits/stdc++. h> is a header file. This file includes **all standard library**.

In programming contests, using this file is a good idea, when you want to reduce the time wasted in doing chores .

So from now onwards You have to use only <bits/stdc++.h> header file always while coding in C++.

```
#include<bits/stdc++.h>
using namespace std;

int main() {
 cout<<sqrt(9);
  return 0;
}
```

if we use **<iostream>** header file, we have to write **<cmath>** header file to run the **sqrt()** function otherwise compiler shows that '*sqrt*' was not declared in this scope but using <bits/stdc++.h> it runs perfectly.