Date: / /
Page:

Day-72

Recursion

* A function which calls itself again & again until a specific condition ~~itself~~ met.

=) Suppose, you want to print —
5 days left for birthday
4 " " " "
3 " " " "
2 " " " "
1 " " " "
Happy Birthday.

=) Then, we can do this —

```
for( int i = 5 ; i > 0 ; i--) {
    cout << i << ": " << "days left for birthday";
    cout << endl;
}
cout << " Happy Birthday ";
```

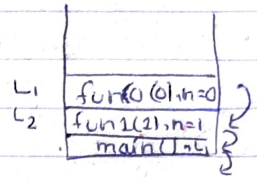
=) This is iterative approach.

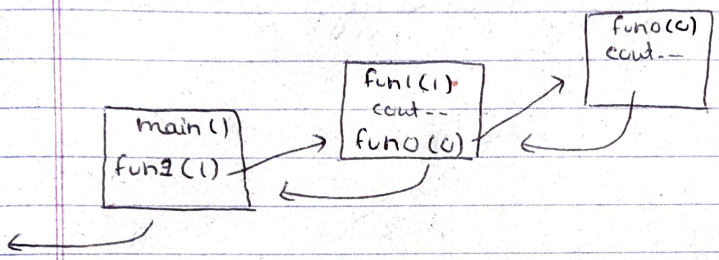
=) Now, we have to solve this problem by using functions.

```
void fun1 (int n){
    cout << n << " days left for birthday";
    fun0 (n-1);
}
void fun0 ( int n ){
    cout << "Happy Birthday";
}
```

Date: / /
Page:

1.
```
int main(){
    fun1(1);
}
```





⇒ What if we do —

```
void fun1( int n){
    cout << n << " days for Birthday ";
    fun1(n-1);
}
```

⇒
```
void fun1 (int n){
    if n==0){
        cout << " Happy Birthday "; } stoppage condition
        return ;
    cout << n << " days left for birthday ";
    fun1(n-1); → Function calling.
}
```

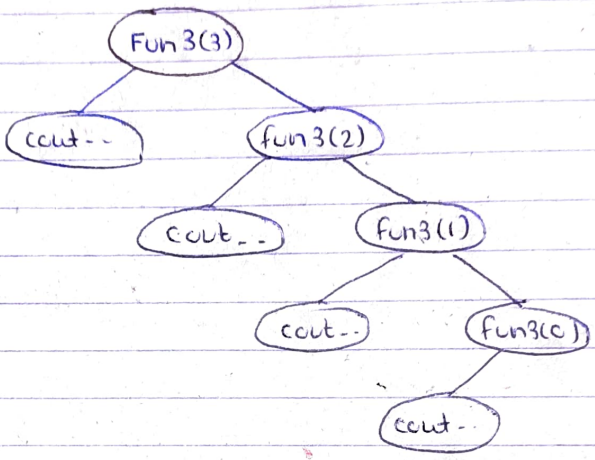
⇒ This is recursion.

⇒ If a problem can be solved by iterative approach then it can also be solved by Recursion.

⇒ When we don't give base condition then stack overflow occur. That is why stoppage condition is necessary.

Date: / /
Page:

Recursion Tree



⇒ **Mathematical way**

Base Condition

Print (0) = Happy Birthday
Print (1) = 1 days left for birthday, Print(0)
Print (2) = 2 " " " " , Print(1)
Print (3) = 3 " " " " , Print(2)
⋮
Print (n) = n days " " " , Print(n-1)
We have to handle this only.

Print n to 1

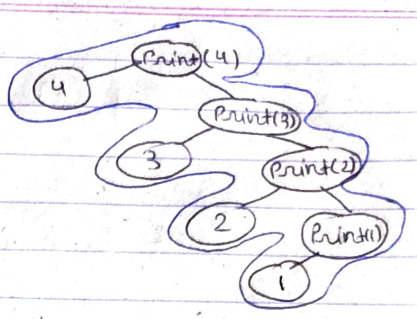
```
for (i= n; i>=1; i--)
    cout << i;
```

} Iterative Approach

⇒
Print(1) = 1
Print(2) = 2, Print(1)
Print(3) = 3, Print(2)
⋮
Print(n) = n, Print(n-1)

```
void Print (int n ){
    if(n==1 ){
        cout<<1;
        return
    }
    cout<<n;
    Print(n-1);
}
```

Recursive Approach



=1 Print n to 1 (Even no. only)

↳ even no.

⇉

Print(2) = 2

Print (4) = 4, Print(2)

Print (6) = 6, Print(4)

⋮

Print(n) = n, Print(n-2)