Recursion

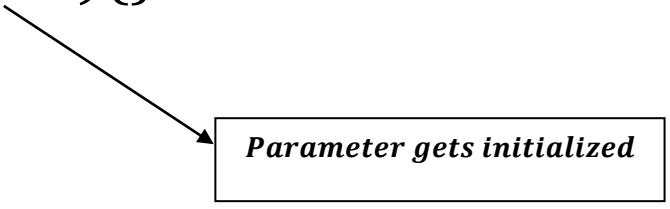
```cpp
#include<iostream>
using namespace std;

void fun(int n){
   if(n==0){
       return;
   }
   int a=2;
   int add=a+n;
   cout << add << endl;
   fun(n - 1);
}

int main(){
    int x=3;
    fun(x);
    return 0;
}
```

1) *initialization*:

$$void\ fun(int\ n)\{\}$$



Initialization **:**

The variables in the form of arguments are passed on to the function.

2)*Decision* :

```
if(n == 0){
    return;
}
```

Decision:

It is also known as base case ,where argument values of decision statement determine whether further recursive calls are required.

3)*Computation* :

$int\ a = 2;$

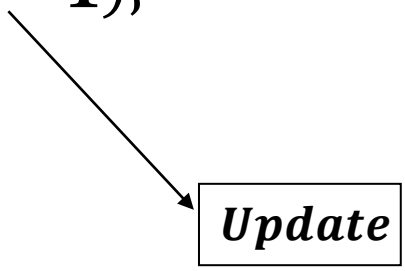
$int\ add = a + n;$

***Computation*:**

The necessary computation is performed using the local variables and the parameters at the current depth.

4)*Update* :

$fun(n - 1);$



Update*: *The update is done so that the variables can be used for further recursive calls.

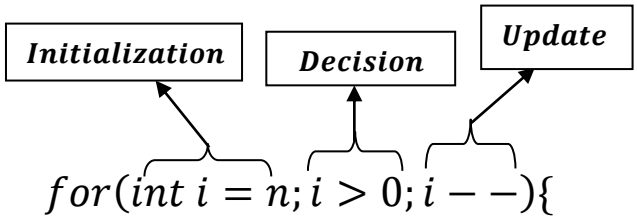
Iteration

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

int main()
{
    int n = 3;
    int a = 2;

    for (int i = n; i > 0; i--)
    {
        int add = a + i;
        cout << add << endl;
    }

    return 0;
}
```



$int\ add = a + i; \rightarrow$ ***Computation***

}

1. *Initialization*: *The decision parameter is set to an initial value, or more precisely, the loop variable is initialized. Other predefined variables are also initialized.*

2. *Decision*: *The decision parameter is used to determine whether further looping is necessary. The loop variable is compared, and based on the outcome, the loop is executed again. The test performed on the loop variable may not be the only condition; other relational expression may also participate in the decision.*

3. *Computation*: *The necessary computation is performed within the loop.*

4. *Update* : *The decision parameter is updated and a transfer is made to the next iteration. The loop variable is incremented /decrement. It may also result in a set of variables being modified.*

Notes on Recursion:

- **Recursive cases have two types of cases ,**

  1)Recursive Cases.

  2)Base Cases.

- ***Every recursive function case must terminate at a base case.***

- ***Generally, iterative solutions are more efficient than recursive solutions*[*due to overhead of function calls*] .

- ***The problems that can be solved recursively can also be solved iteratively.***

- ***Recursion exists because of the reursive nature of the process, or because of the recursive structure of the data to be processed.***

  For some cases , the recursive solution may be simpler than iteration.

For recursion, additional amount of work must be done in the form of saving return addresses, so that correct statement is executed next, when the return is executed.

The function should also have formal parameters and local variables onto the stack upon entry and restore these parameters and variables on completion.

But if the problem itself is recursive in nature, an iterative solution will have to simulate a stack. In such cases, recursion may turn out to be more efficient.

****