**RECURSION**

**Recursion** is a method of solving problems that involves breaking down a problem into smaller sub problems until you get to a small enough problem that can be solved trivially. Usually recursion involves a function calling itself.

**EXAMPLE 1:**

void f1(int n)

{

if(n>0)

{

printf("\n%d",n);

**f1(n-1);**

}

}

void main()

{

int a=3;

f1(a);

}

**EXAMPLE 2:**

void f2(int n)

{

if(n>0)

{

**f2(n-1);**

printf("\n%d",n);

}

}

void main()

{

int a=3;

f2(a);

}

**UNDERSTANDING CALLING & RETURNING TIME**

void fun(int n)

{

if( <some \_cond>)

{

.............. // This code will be executed before recursive call and so it's called **CALLING TIME EXECUTION**

fun(n-1);

.............. // This code will be executed after recursive call and so it's called **RETURNING TIME EXECUTION**

}

}

**RECURSION V/s ITERATION**

|  |  |
| --- | --- |
| **RECURSION** | **ITERATION** |
| 1. Used for repetitive kinds of tasks | 1. It is also used for repetitive kinds of tasks |
| 2. Contains both calling time & path returning time execution path | 2. Contains only calling time execution |
| 3. Heavily uses stack | 3.Uses stack very mildly |
| 4. Creates multiple copies of the local var and parameter, one copy for each call | 4. Creates only one copy of local var & parameters |
| 5. Generally Slow in speed | 5. Generally faster in speed |
| 6. Reduces the length of the code | 6. Increases the length of the code |

**HOW RECURSION USES STACK ?**

void f1(int n)

**Main Memory**

{

**Heap Area**

if(n>0)

{

printf("\n%d",n);

**f1(n-1);**

}

}

**Stack Area**

void main()

{

int a=3;

**Code Segment**

f1(a);

}

int fun(int n)

**Main Memory**

{

**Heap Area**

if(n>0)

{

return **fun(n-1)+n;**

}

return 0;

}

**Stack Area**

void main()

{

int a=5;

**Code Segment**

print("%d",fun(a));

}

**OUTPUT: 15**

**TYPES OF RECURSION**

1. Tail Recursion

2. Head Recursion

3. Tree Recursion

4. Indirect Recursion

5. Nested Recursion

**TAIL RECURSION:**

If the recursive call of the function is the last statement in it's body , then it is called tail recursion

**EXAMPLE:**

void f1(int n)

{

if(n>0)

{

printf("\n%d",n);

**f1(n-1);**

}

}

void main()

{

int a=3;

**f1(a);**

}

**HEAD RECURSION:**

If the recursive call of the function is the first statement in it's body , then it is called head recursion. In other words if the function is not executing any statements during calling time , and performing all the operations in returning time , then it is called head recursion.

**EXAMPLE:**

void f2(int n)

{

if(n>0)

{

**f2(n-1);**

printf("\n%d",n);

}

}

void main()

{

int a=3;

**f2(a);**

}

**TREE RECURSION:**

If a function is calling itself recursively more than one time in its body then , then it is called tree recursion. In other words if the function contains 2 or more calls to itself then it is called tree recursion

**EXAMPLE:**

void fun(int n)

{

if(n>0)

{

printf("%d,",n);

**fun(n-1);**

**fun(n-1);**

}

}

void main()

{

int a=3;

**fun(a);**

}

**OUTPUT:**

3 ,2 ,1 ,1 ,2 ,1 ,1 ,

**INDIRECT RECURSION:**

If 2 or more functions are calling themselves in a CIRCULAR MANNER , then it is called indirect recursion.

**EXAMPLE:**

void funA(int n)

{

if(n>0)

{

printf("%d,",n);

**funB(n-1);**

}

}

void funB(int n)

{

if(n>0)

{

printf("%d,",n);

**funA(n/2);**

}

}

void main()

{

int x=20;

**funA(x);**

}

**OUTPUT:**

20,19,9,8,4,3,1

**NESTED RECURSION:**

If the recursive call of a function calls itself in its parameter then it is called nested recursion recursion. In other words if the call to the same function is present in the parameter list of a recursive call then it is called nested recursion

**EXAMPLE:**

int fun(int n)

{

if(n>100)

{

return n-10;

}

else

{

return **fun(fun(n+11));**

}

}

void main()

{

int a=95;

printf("%d",**fun(a)**);

}

**OUTPUT:**

91