

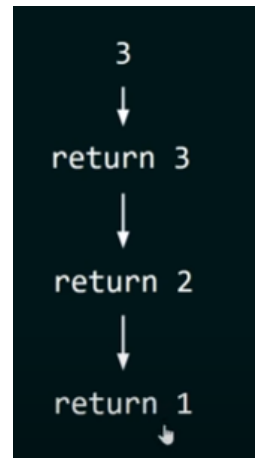
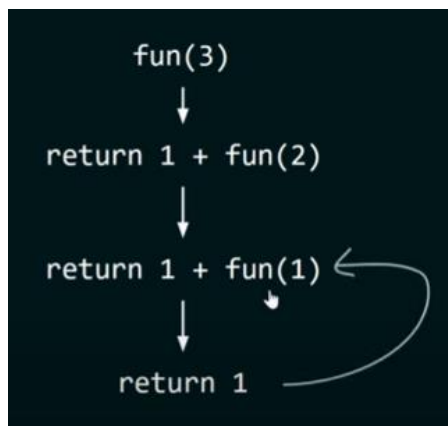
- Recursion is a process in which a function calls itself directly or indirectly.

```
#include <stdio.h>

int fun(int n) //caller
{
    if (n == 1)
        return 1;
    else
        return 1 + fun(n - 1); //calling
}

int main()
{
    int n = 3;
    printf("%d", fun(n));
}
```

Output: 3



- Determine the factorial of a number.

```
#include <stdio.h>

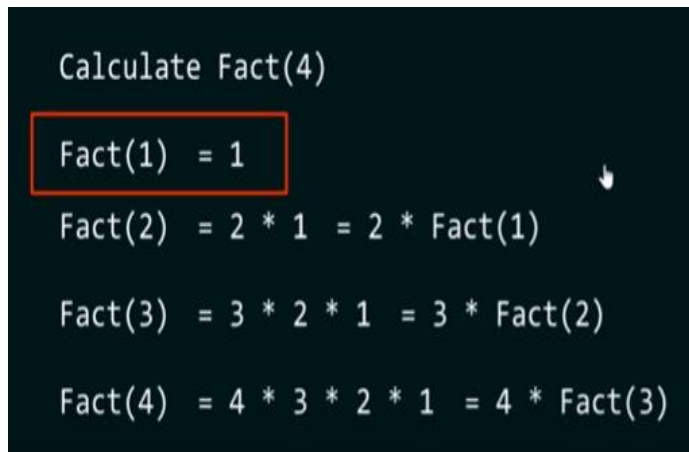
int fact(int n)
{
    if (n == 1)
        return 1;
    else
        return n * fact(n - 1);
}

int main()
{
    int n;
    printf("Enter the number : ");
    scanf("%d", &n);
    printf("Factorial of %d is %d\n", n, fact(n));
}
```

Output:

Enter the number : 4
Factorial of 4 is 24

Fact(n) = n * fact(n-1)



```
Calculate Fact(4)
Fact(1) = 1
Fact(2) = 2 * 1 = 2 * Fact(1)
Fact(3) = 3 * 2 * 1 = 3 * Fact(2)
Fact(4) = 4 * 3 * 2 * 1 = 4 * Fact(3)
```

- WAP to print numbers from 1 to 10 in such a way that when number is odd, add 1 and when number is even, subtract 1.

```
#include <stdio.h>
void odd();
void even();
int n = 1;

void odd()
{
    if (n <= 10) {
        printf("%d ", n + 1);
        n++;
        even();
    }
    //return;
}
void even()
{
    if (n <= 10) {
        printf("%d ", n - 1);
        n++;
        odd();
    }
    //return;
}
int main()
{
    odd();
}
```

Output:

2 1 4 3 6 5 8 7 10 9

TYPES OF RECURSION

- 1 Direct recursion
- 2 Indirect recursion
- 3 Tail recursion
- 4 Non-tail recursion

1 Direct recursion

A function is called direct recursive if it calls the same function again.

Structure of Direct recursion:

```
fun(){
    //some code

    fun();

    //some code
}
```

2 Indirect recursion

A function (let say fun) is called indirect recursive if it calls another function (let say fun2) and then fun2 calls fun directly or indirectly.

Structure of Indirect recursion:

```
fun() {
    //some code
    fun2();
    //some code
}

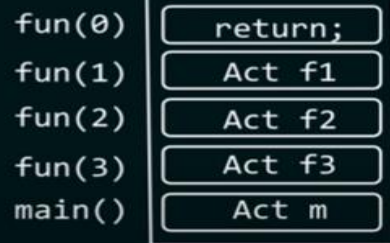
fun2() {
    //some code
    fun();
    //some code
}
```

Tail recursive :

DEFINITION

A recursive function is said to be **tail recursive** if the recursive call is the last thing done by the function. There is no need to keep record of the previous state.

```
void fun(int n) {  
    if(n == 0)  
        return;  
    else  
        printf("%d ", n);  
    return fun(n-1);  
}  
int main() {  
    fun(3);  
    return 0;  
}
```



Output: 3 2 1

Non-tail recursive :

DEFINITION

A recursive function is said to be **non-tail recursive** if the recursive call is not the last thing done by the function. After returning back, there is some something left to evaluate.

```
void fun(int n) {  
    if(n == 0)  
        return;  
    fun(n-1);  
    printf("%d ", n);  
}  
int main() {  
    fun(3);  
    return 0;  
}
```

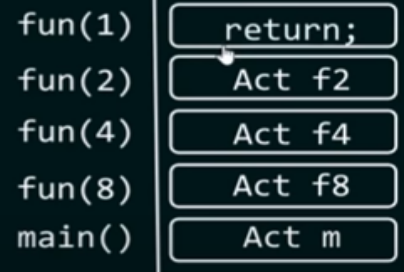


Output: 1 2 3

One more example of non-tail function.

ONE MORE EXAMPLE (NON-TAIL)

```
int fun(int n) {  
    if(n == 1)  
        return 0;  
    else  
        return 1 + fun(n/2);  
}  
int main() {  
    printf("%d", fun(8));  
    return 0;  
}
```



The output of this program is 3

$$1 + 0 = 1$$

$$1 + 1 = 2$$

$$1 + 2 = 3$$

This 3 return to the main. That's why the output is 3

!(1). Local and Global Variable.

```
#include <stdio.h>
int x; /*Global Variable*/
int main()
{
    int y; /*Local Variable*/

    printf("x = %d, y = %d\n", x, y);
}
```

Output:
x = 0, y = 0

Remember:

- ফাংশনের প্যারামিটার হিসাবে যেসব ভেরিয়েবল পাঠানো হয় সেগুলো লোকাল ভেরিয়েবল।
- Global Variable declare করার সময় কোনো মান অ্যাসাইন না করলে অটোমেটিক ০ প্রিন্ট হয়।

1(2). Local and Global Variable.

```
#include <stdio.h>
int x = 1;
void myfunction(int y)
{
    y = y * 2;
    x = x + 10;
    printf("Void function\n");
    printf("x = %d, y = %d\n", x, y);
}
int main()
{
    int y = 5;
    x = 10;

    myfunction(y);
    printf("Main function\n");
    printf("x = %d, y = %d\n", x, y);
}
```

Output:
Void function
x = 20, y = 10
Main function
x = 20, y = 5

প্রোগ্রাম টা একটু মনোযোগ দিয়ে দেখো তাহলেই বুঝতে পারবা।

Remember:

- এখানে 2টি y-ই আলাদা, তারা দুটি আলাদা ফাংশনের লোকাল ভেরিয়েবল।
একটি ফাংশনের লোকাল ভেরিয়েবল আরেকটি ফাংশনের ভেরত থেকে এক্সেস করা যায় না।
- x যেহেতু গ্লোবাল ভেরিয়েবল তাই মেইন ফাংশনে x এর মান 20 প্রিন্ট হবে।

2(1). Static variable

```
#include <stdio.h>
int a;
static int b;
void function()
{
    a = a + 1;
    b = b + 1;
}
int main()
{
    function();
    printf("a = %d\n", a);
    printf("b = %d\n", b);
}
```

Output:

a = 1
b = 1

Remember:

a and b কে গ্লোবাল স্পেস এ ডিক্লেয়ার করা হয়েছে। তাই এদেরকে ইনিশিয়ালাইজড না করা সত্ত্বেও মান ০ হয়ে গেছে। এরপর ফাংশনের মধ্যে এদের মান ১ করে বাড়িয়ে দিলাম।

Global variable: যেকোনো ফাইলের যেকোনো ফাংশন থেকে এক্সেস করা যাবে।

Static global variable: শুধু মাত্র যে ফাইলে ডিক্লেয়ার করা হয়েছে ওই ফাইলের ফাংশন গুলোর মধ্যে এক্সেস করা যাবে।

2(2). Static variable

```
#include <stdio.h>
void function()
{
    int a = 10;
    static int b = 10;

    a = a + 2;
    b = b + 2;
    printf("a = %d, b = %d\n", a, b);
}
int main()
{
    function();
    function();
    function();
}
```

Output:

a = 12, b = 12
a = 12, b = 14
a = 12, b = 16

b is **static int** variable.
a always print 12 but b print 12, 14, 16
This the rule of **static** variable.

3(1). Recursion basic problem.

```
#include <stdio.h>
void recursion(int count)
{
    if (count == 5)
    {
        return;
    }
    printf("I am learning recursion\n");
    recursion(count + 1);
    return;
}
int main()
{
    recursion(1);
}
```

Output:

```
I am learning recursion
I am learning recursion
I am learning recursion
I am learning recursion
```

Recursion হল একটি ফাংশন নিজেই নিজেকে কল করা। তবে কল করার কাজটি বুঝে-শুনে করতে হবে।
কখন recursion বন্ধ করতে হবে সেটি প্রোগ্রাম এর ভিতর বলে দিতে হবে।

3(2). Recursion basic problem.

```
#include <stdio.h>
void recursion(int count)
{
    if (count > 5)
    {
        return;
    }
    printf("Count = %d\n", count);
    recursion(count + 1);
}
int main()
{
    recursion(1);
}
```

Output:

```
Count = 1
Count = 2
Count = 3
Count = 4
Count = 5
```


3(3). Recursion basic problem.

```
#include <stdio.h>
void recursion(int count)
{
    if (count > 5)
    {
        return;
    }
    recursion(count + 1);
    printf("Count = %d\n", count);
}
int main()
{
    recursion(1);
}
```

Output:
Count = 5
Count = 4
Count = 3
Count = 2
Count = 1

কোনো ফাংশনের return type যদি void হয় তাহলে ফাংশনের শেষ লাইনে return; না লিখলেও চলে। আপনা আপনি সেটি রিটার্ন করে। void মানে কম্পাইলার তার কাছ থেকে কোনো কিছু আশা করে না। সে ঠিক ঠাক ফেরত আসলেই খুশি।

3(4). Recursion basic problem.

```
#include <stdio.h>
void recursion(int count)
{
    if (count > 5)
    {
        return;
    }
    printf("Count = %d\n", count);
    recursion(count + 1);
    printf("Count = %d\n", count);
}
int main()
{
    recursion(1);
}
```

Output:
Count = 1
Count = 2
Count = 3
Count = 4
Count = 5
Count = 5
Count = 4
Count = 3
Count = 2
Count = 1

3(5). Recursion basic problem.

```
#include <stdio.h>
void recursion()
{
    static int count = 1;
    if (count > 5)
    {
        return;
    }
    printf("Count = %d\n", count);
    count = count + 1;
    recursion();
}
int main()
{
    recursion();
}
```

Output:
Count = 1
Count = 2
Count = 3
Count = 4
Count = 5

Static Variable ব্যবহার করেই ফাংশনটি কতবার চলবে তা নিয়ন্ত্রণ করতে পারি।
সেক্ষেত্রে আর count নামক ভেরিয়েবল আমাদের ফাংশনের প্যারামিটার হিসাবে বারবার পাঠাতে হবে না।

4(1). Factorial using recursion

```
#include <stdio.h>
int factorial(int n)
{
    if (n == 0)
        return 1;
    else
        return n * factorial(n - 1);
}
int main()
{
    int n;
    scanf("%d", &n);

    if (n < 0)
    {
        printf("Undefined\n");
        //return 0;
    }
    printf("Factorial of %d is %d\n", n, factorial(n));
}
```

Output:
5
Factorial of 5 is 120

4(2). Factorial using recursion

```
#include <stdio.h>
#include <string.h>

int call = 0;
int factorial(int n)
{
    call = call + 1;
    if (n == 0)
        return 1;
    else
        return n * factorial(n - 1);
}
int main()
{
    int n;
    scanf("%d", &n);

    if (n < 0)
    {
        printf("Undefined\n");
        //return 0;
    }
    printf("Factorial of %d is %d\n", n, factorial(n));
    printf("Number of fucntion calls : %d\n", call);
}
```

Output:
5
Factorial of 5 is 120
Number of fucntion calls : 6

Factorial function-টি কতবার কল করা হয়েছে সেটি বের করার প্রোগ্রাম।
এটা বের করার জন্য একটি গ্লোবাল ভেরিয়েবল প্রয়োজন।

5(1). Fibonacci number

```
#include <stdio.h>
#include <string.h>

int call = 0;
int fibo(int n)
{
    call = call + 1;
    if (n == 1 || n == 2)
        return 1;
    else
        return fibo(n - 1) + fibo(n - 2);
}
int main()
{
    int n;
    char str[3];
    scanf("%d", &n);

    if (n == 1) strcpy(str, "st");
    else if (n == 2) strcpy(str, "nd");
    else if (n == 3) strcpy(str, "rd");
    else strcpy(str, "th");

    printf("%d %s fibonacci number is = %d\n", n, str, fibo(n));
    printf("Number of function calls = %d\n", call);
}
```

Output:

```
6
6 th fibonacci number is = 8
Number of function calls = 15
```

5(2). Fibonacci number

```
#include <stdio.h>
#include <string.h>

int call = 0;
int fib[50];
int fibo(int n)
{
    call = call + 1;
    if (fib[n] != 0)
        return fib[n];

    if (n == 1 || n == 2)
        return fib[n];
    else
        return fib[n] = fibo(n - 1) + fibo(n - 2);

    return fib[n];
}
int main()
{
    int n;
    char str[3];

    fib[1] = 1;
    fib[2] = 1;

    scanf("%d", &n);

    if (n == 1) strcpy(str, "st");
    else if (n == 2) strcpy(str, "nd");
    else if (n == 3) strcpy(str, "rd");
    else strcpy(str, "th");

    printf("%d %s fibonacci number is = %d\n", n, str, fibo(n));
    printf("Number of function calls = %d\n", call);

    for (n = 1; n < 12; n++) {
        printf("%2d : %2d\n", n, fib[n]);
    }
}
```

Output:

```
5
5 th fibonacci number is = 5
Number of function calls = 7
1 : 1
2 : 1
3 : 2
4 : 3
5 : 5
6 : 0
7 : 0
8 : 0
9 : 0
10 : 0
11 : 0
```