

RECURSION / RECURSIVE FUNCTIONS

It is the process of calling a function itself.

Purpose:

Recursion allows the user to get results , without using loops. Due to this complexity of program is reduced.

Recursion reduce calling of function by the user.

By using recursion, we can control the function calling information or statements.

By using recursion, we can evaluate stack expressions.

Drawbacks:

They are slower than normal functions due to stack over lapping.

They can create stack over flow because of occupying more stack.

Recursion functions will create infinitive loops also.

Eg: 1

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

```
#include<conio.h>
```

```
void main()
{
printf("Welcome to C\n");
main();
}
```

Note: This program causes infinitive loops.

Eg 2: Controlling the above program

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

int  a=1; /* global variable*/

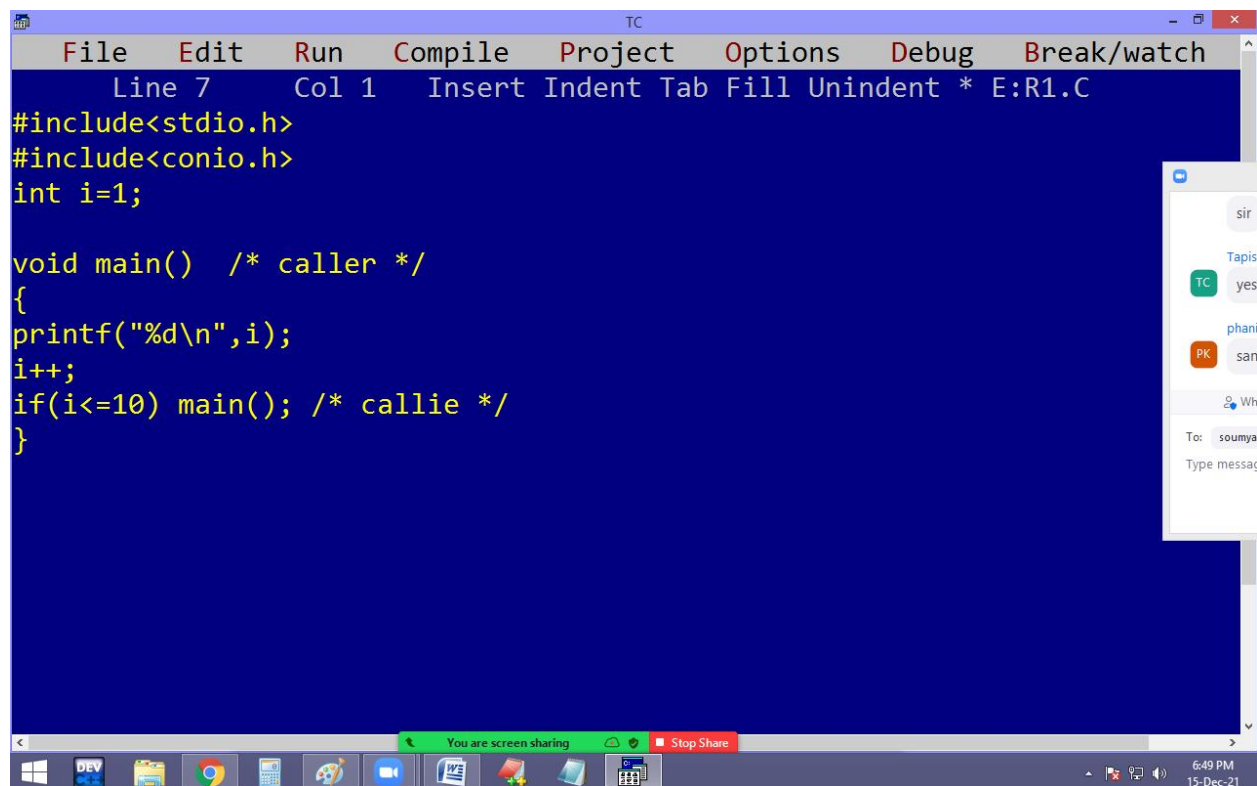
void main()
{
printf("Welcome to C\n");

a++;

if(a<=3) main();

getch();
}
```

Eg. printing 1..10 numbers using recursion.

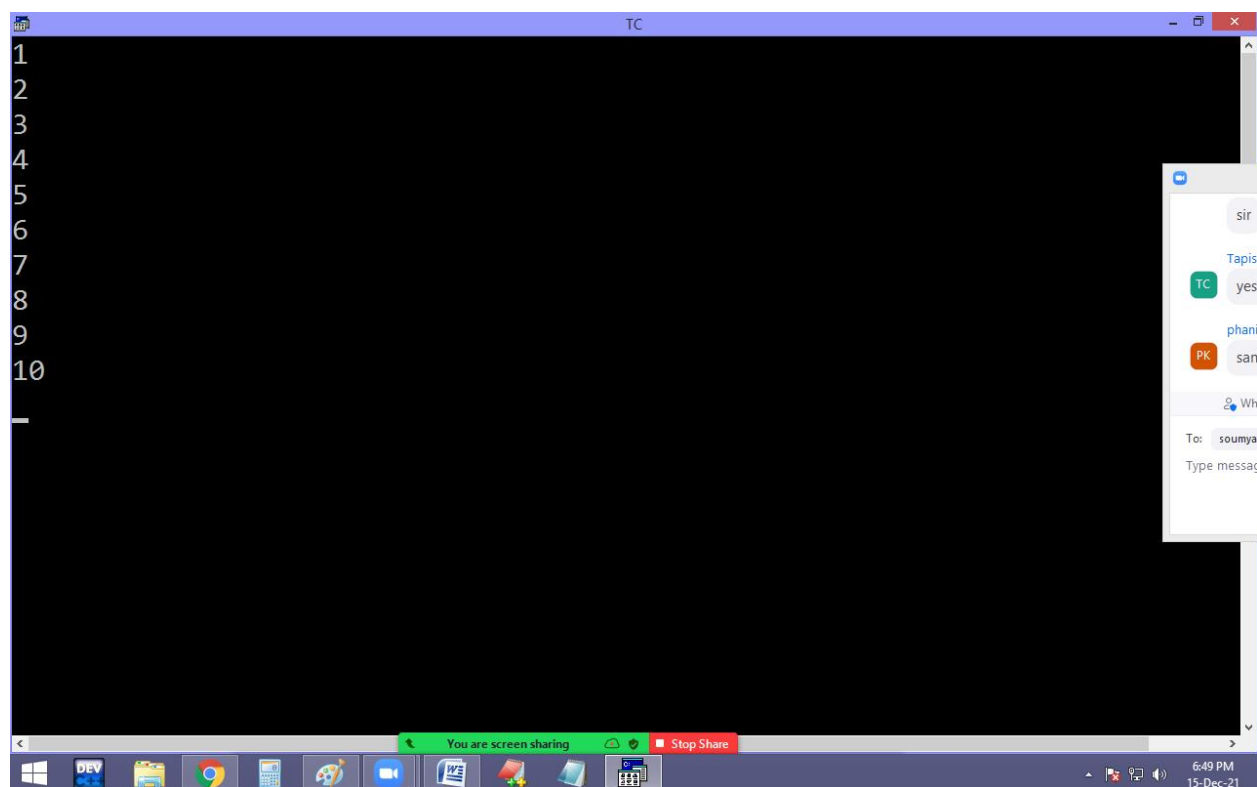


The screenshot shows the Turbo C++ (TC) IDE with a blue background. The menu bar includes File, Edit, Run, Compile, Project, Options, Debug, and Break/watch. The status bar at the top indicates 'Line 7 Col 1 Insert Indent Tab Fill Unindent * E:R1.C'. The code in the editor is as follows:

```
#include<stdio.h>
#include<conio.h>
int i=1;

void main() /* caller */
{
printf("%d\n",i);
i++;
if(i<=10) main(); /* callie */
}
```

On the right side, there is a chat window with messages from 'sir', 'Tapis', 'yes', 'phani', 'PK', 'sarr', and 'Wh'. The bottom status bar shows 'You are screen sharing' and 'Stop Share' buttons. The taskbar at the bottom includes icons for Windows, DEV, File Explorer, Chrome, Calculator, Paint, and other applications. The system clock shows 6:49 PM on 15-Dec-21.



The screenshot shows the Turbo C++ (TC) IDE with a black background. The output window on the left displays the numbers 1 through 10, each on a new line. The chat window on the right is the same as in the previous screenshot. The bottom status bar and taskbar are also the same.

Eg: Finding factorial using recursion:

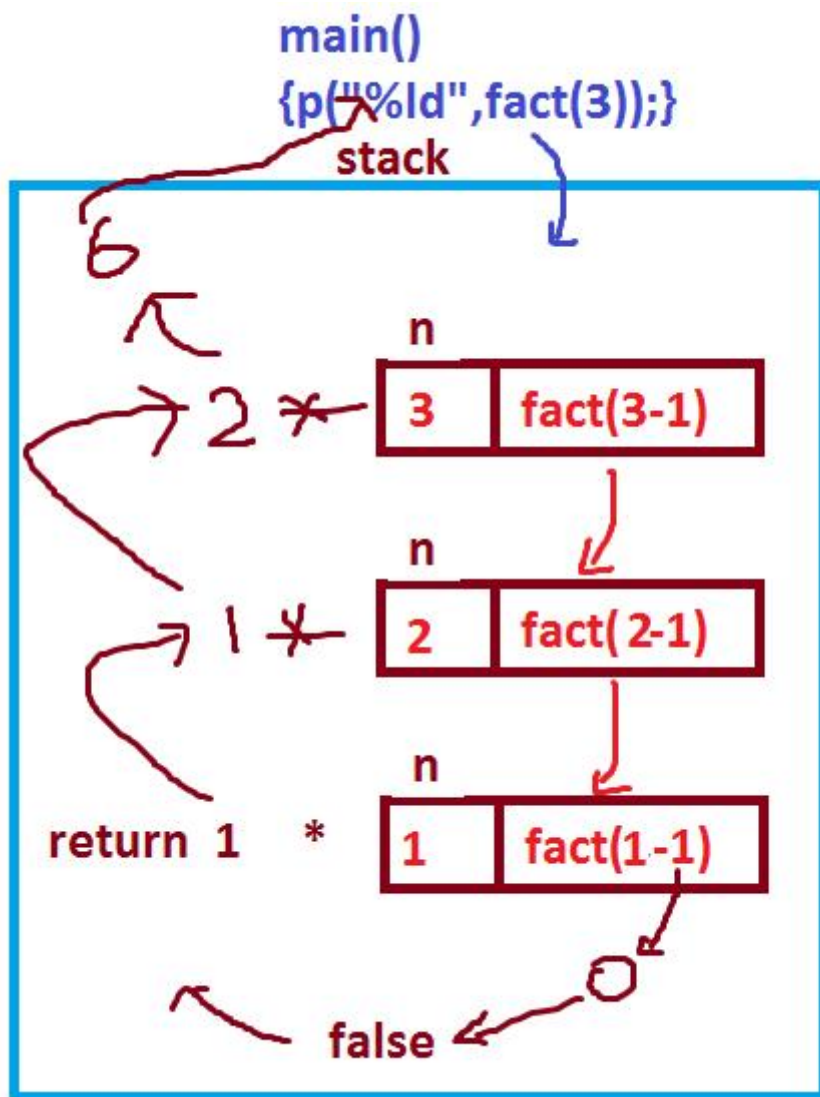
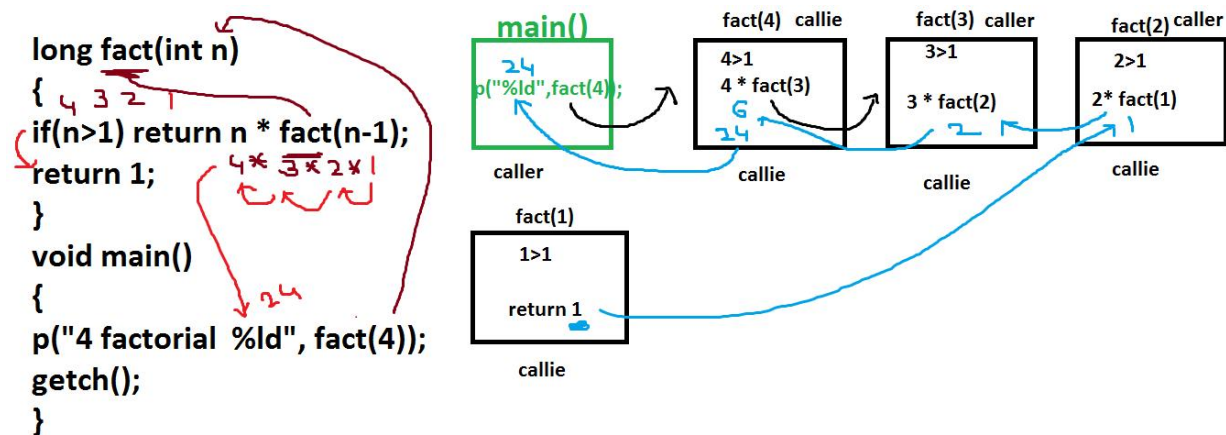
```
#include<stdio.h>
#include<conio.h>

long fact(int n)
{
    if(n!=0) return n * fact(n-1); else return 1;
}

void main()
{
    int n;
    clrscr();
    printf("Enter a no "); scanf("%d", &n);
    printf("%d Factorial = %ld", n, fact(n));
    getch();
}
```

O/P: Enter a no 5

5 Factorial = 120

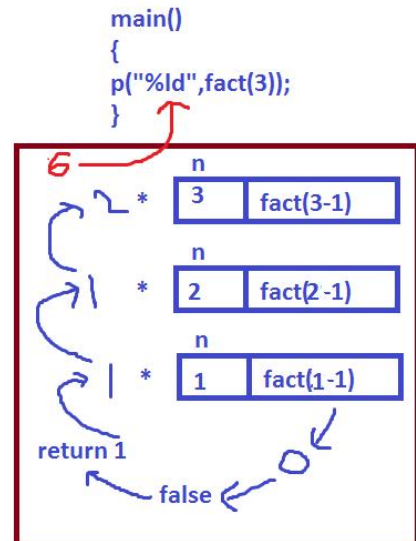


```

#include<stdio.h>
#include<conio.h>
long fact(int n)
{
if(n!=0) return n * fact(n-1);
else return 1;
}
void main() /* main is caller */
{
printf("3 factorial=%ld",fact(3));
getch();
}

```

6=3 * 2 * 1 * 1



Finding power using recursion:

```

#include<stdio.h>

```

```

#include<conio.h>

```

```

long power(int b, int p)

```

```

{

```

```

if(p!=0) return b * power(b, p-1); else return 1;

```

```

}

```

```

void main()

```

```

{

```

```

int b,p;

```

```

printf("Enter base, power values ");

```

```
scanf("%d %d",&b,&p);

printf("%d ^ %d = %ld", b, p, power(b,p));

getch();

}
```

Output: Enter base, power values 2 5

$2^5 = 32$

```
#include<stdio.h>
#include<conio.h>
long power(int b, int p)
{
if(p!=0) return b * power(b,p-1);
else return 1;
}
void main() /* main is caller */
{
printf("2^3=%ld",power(2,3));
getch();
}
```

Handwritten notes on a blue background:

$\frac{p}{2}$ $\frac{b}{2}$

$2 = 2 \times 2 \times 2 \times 1$

An arrow points from the handwritten 2 in the multiplication to the $p \neq 0$ condition in the code.

Eg : Finding digital sum using recursion

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

```
#include<conio.h>
```

```
int s=0; /* global var*/
```

```
int dsum(long n)
```



```
{  
if(n!=0)  
{  
s=s+n%10;  
dsum(n/10);  
}  
return s;  
}  
  
void main()  
{  
long n;  
clrscr();  
printf("Enter a no");  
scanf("%ld",&n);  
printf("%ld digital sum = %d",n,dsum(n));  
getch();  
}
```

Output:

Enter a no: 123

123 digital sum = 6

```
#include<stdio.h>
#include<conio.h>
int dsum(long int n)
{
    static int s;
    if(n!=0)
    {
        s+=n%10; dsum(n/10);
    }
    return s;
}
void main() /* main is caller */
{
    printf("123 digital sum=%d",dsum(123));
    getch();
}
```

$$\frac{n}{10}$$

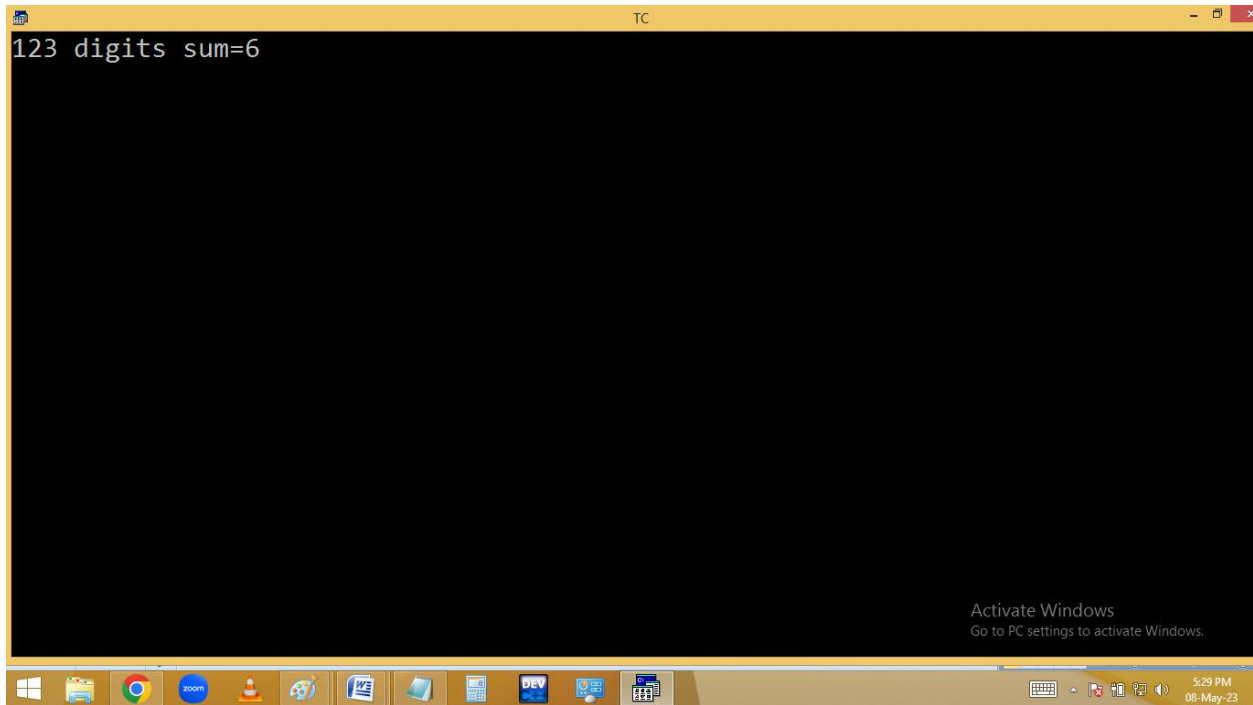
$$\frac{5}{0+3+2+1}$$

6

```
TC
File Edit Run Compile Project Options Debug Break/watch
Line 6 Col 10 Insert Indent Tab Fill Unindent * E:4PM.C
#include<stdio.h>
#include<conio.h>
int dsum(long n)
{
    if(n!=0) return n%10 + dsum(n/10);
    return 0;_
}
void main()
{
    printf("123 digits sum=%d",dsum(123));
    getch();
}
```

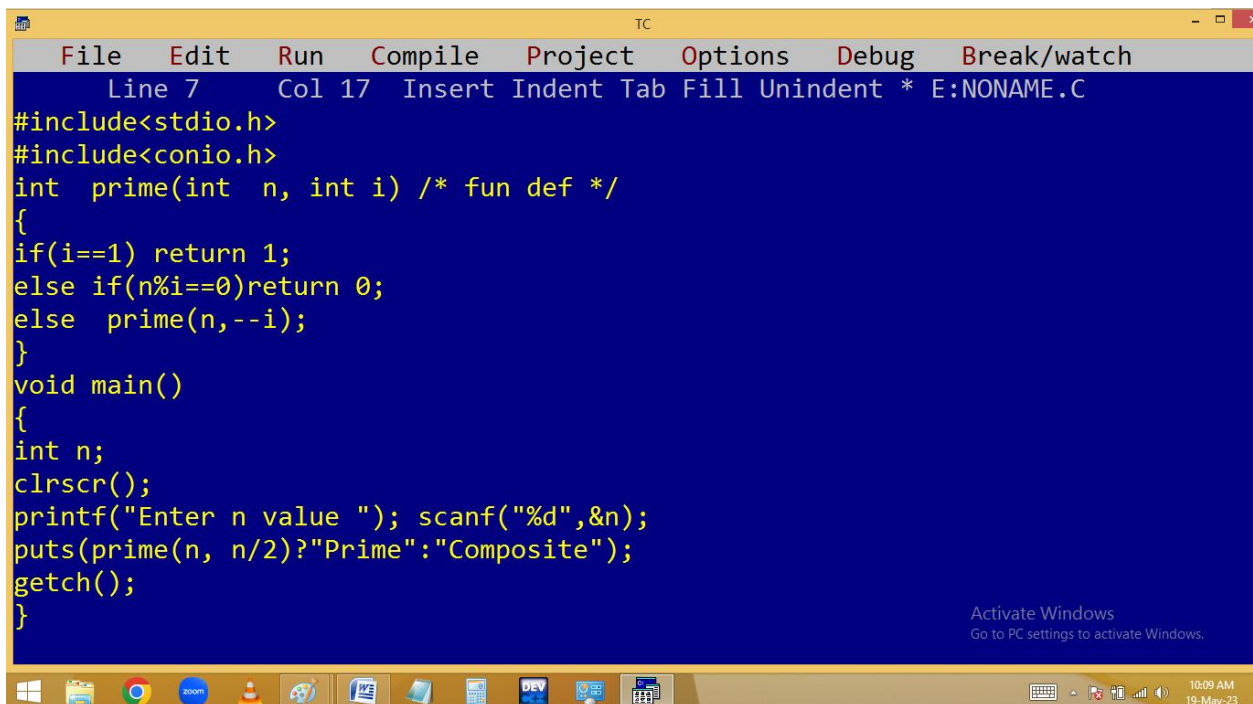
Activate Windows
Go to PC settings to activate Windows.

5:28 PM
08-May-23

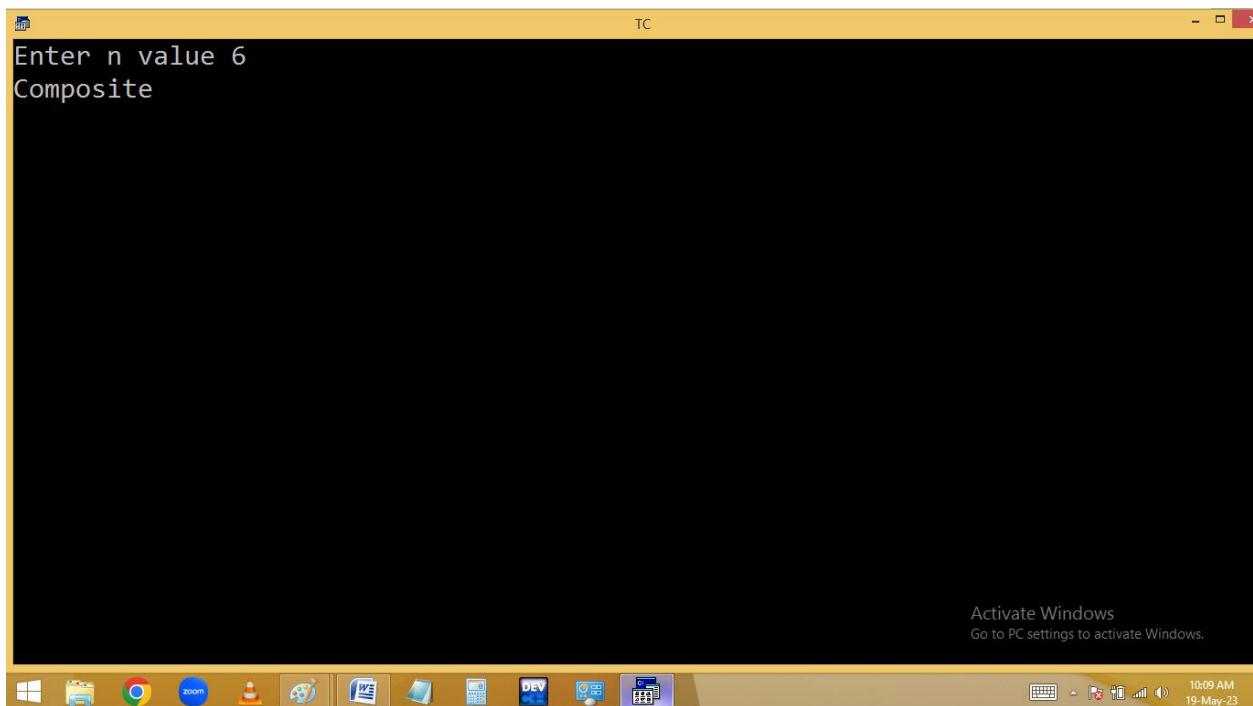
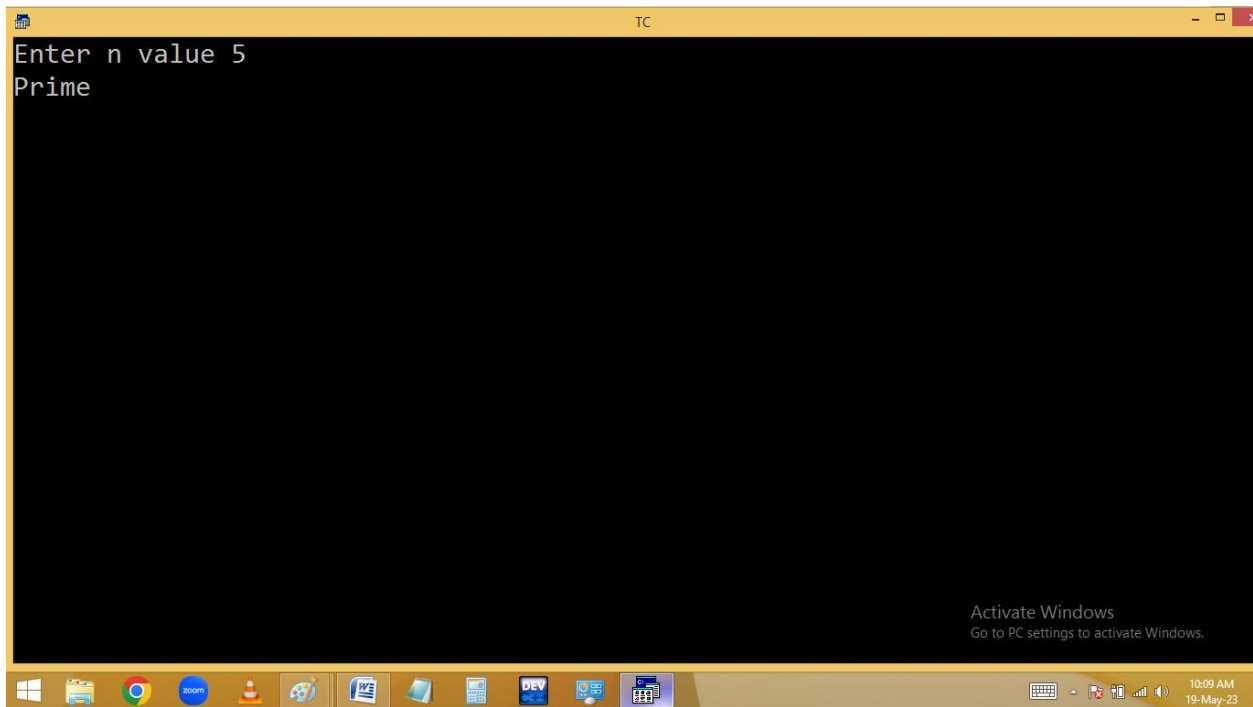


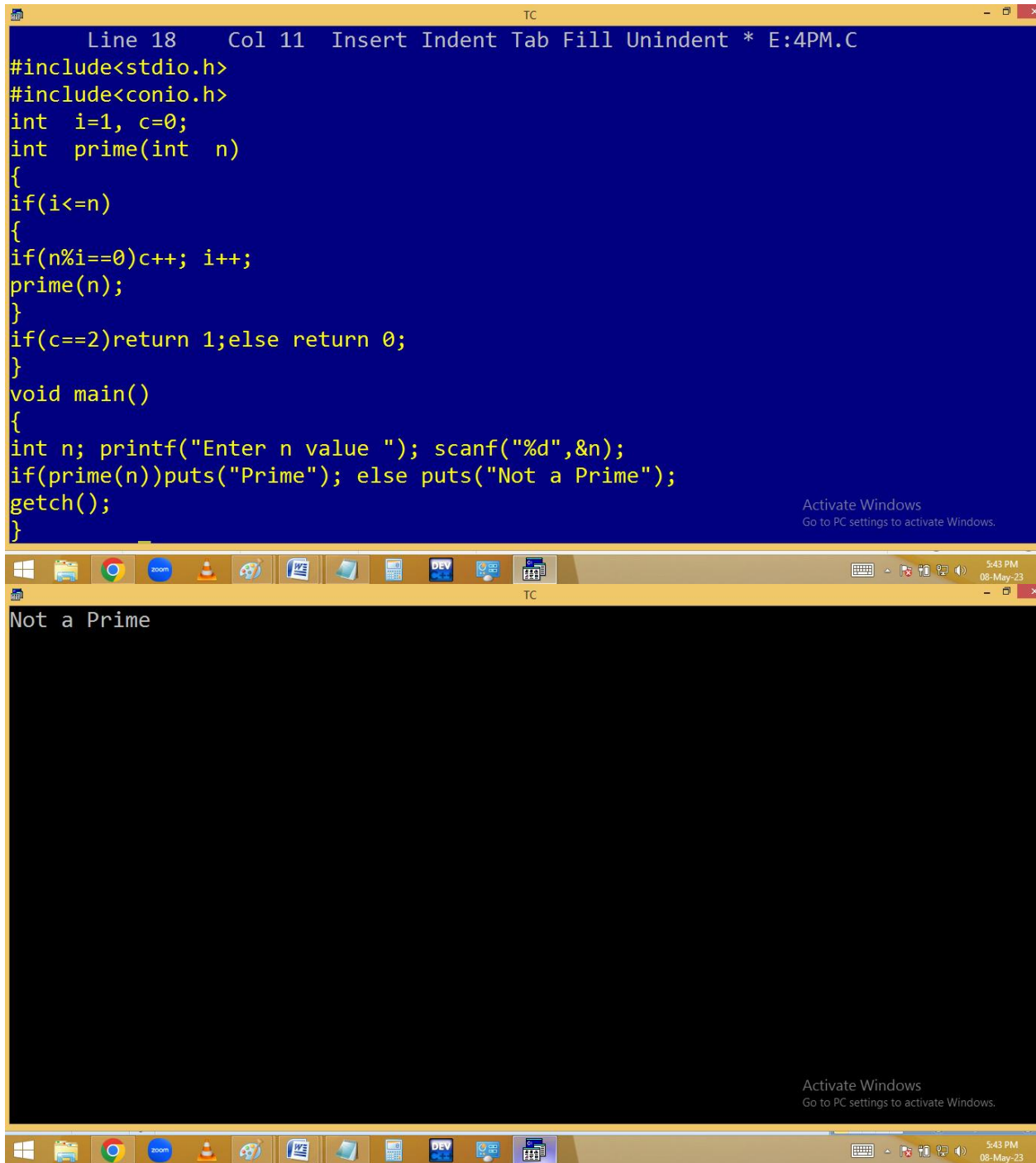
```
123 digits sum=6
```

Finding prime using recursion:



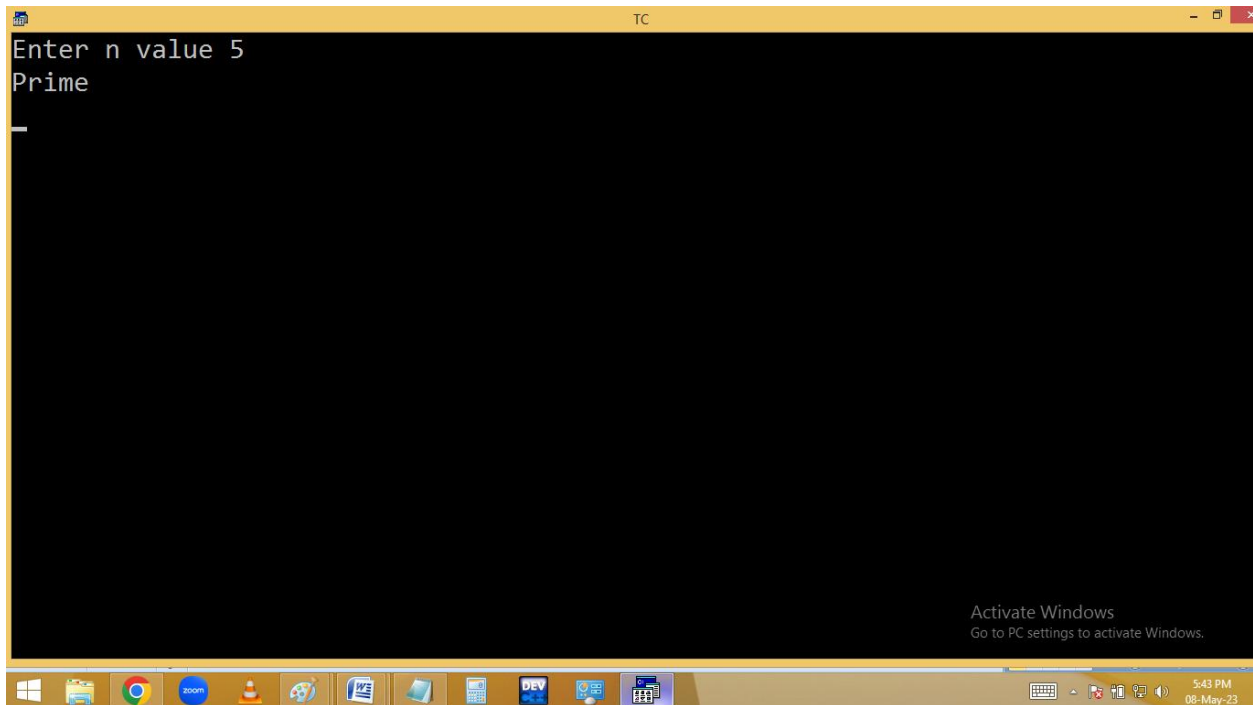
```
File Edit Run Compile Project Options Debug Break/watch
Line 7 Col 17 Insert Indent Tab Fill Unindent * E:NONAME.C
#include<stdio.h>
#include<conio.h>
int prime(int n, int i) /* fun def */
{
    if(i==1) return 1;
    else if(n%i==0) return 0;
    else prime(n,--i);
}
void main()
{
    int n;
    clrscr();
    printf("Enter n value "); scanf("%d",&n);
    puts(prime(n, n/2)?"Prime":"Composite");
    getch();
}
```





```
Line 18   Col 11   Insert Indent Tab Fill Unindent * E:4PM.C
#include<stdio.h>
#include<conio.h>
int  i=1, c=0;
int  prime(int  n)
{
if(i<=n)
{
if(n%i==0)c++; i++;
prime(n);
}
if(c==2)return 1;else return 0;
}
void main()
{
int n; printf("Enter n value "); scanf("%d",&n);
if(prime(n))puts("Prime"); else puts("Not a Prime");
getch();
}
```

Not a Prime



```
Enter n value 5
Prime
```

STORAGE CLASSES

Storage classes decides the behavior/ properties of a variable like

1. **Storage**: In which memory area the variables are stored.
2. **Initial values**: Without initialization, the variable initial (first) values.
3. **Scope**: In which blocks the variables are available i.e. where we can use these variables.
4. **Life time**: Until which time these variables are active (live) in memory.

We have 2 types of Storage classes.

1. Automatic storage class: It contain two specifiers
 - a.auto
 - b.register
2. Static storage class: It contain two specifiers.
 - a.static
 - b.extern

auto variables:

By default all the general / local variables are automatic variables and they created/deleted automatically.

Eg: 1

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

void main()
{
    auto int a=100;  /* int a=100; */
    printf("a = %d\n", a);
    getch();
}
```

Output: a = 100

Eg:2

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

```
#include<conio.h>
```

```
void fun()
```

```
{
```

```
auto int a=10;
```

```
}
```

```
void main()
```

```
{
```

```
a=20;
```

```
}
```

Note: This program gives error because of variable 'a' scope is within function only.

Register variables:

They declared with register keyword.

They stored in **CPU registers**.

Their initial values are garbage.

They are faster than auto variables.

Generally the system is having 4 to 8 registers based on CPU capacity.

When several variables are declared as register variables, the excess variables will become auto variables.

Register variables doesn't have any address. Due to this they are not used in pointer and pointer related operations.


Eg: 1

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

void main()
{
    register int a = 100;
    printf("a = %d" ,a);
    getch();
}
```

Eg: 2

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

void main()
{
    register int a = 100;
    printf("a addr is = %u" ,&a);  error
    getch();
}
```

Static variables:

The variables that are declared with static keyword are called static variables.

They stored in static area of **data segment**.

Static variables are created/initialized only once at the first time they have been invoked and it is shared by that function several times.

Static variables initial value is **0**.

Their scope is within block / function only.

They are active in memory until total program is executed / closed [life time].

Eg:

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

void fun()
{
    static int a;
    printf("a = %d\n",++a);
}

void main()
{
    clrscr();
    fun();
    fun();
    fun();
    getch();
}
```

Output:

a=1

a=2

a=3

Extern variables:

The variables that are declared with extern keyword are called extern variables.

They stored in data segment, which is the public area.

initial value of extern variable is **0**.

extern variables declared inside a function and available to below functions also.

extern variable scope is to all functions declared after the variable.

life time is until total program is executed.

Every extern variable has 2 states.

1.Declared / declaration

Eg: extern int a;

2.Defined / definition

Eg: int a; or int a=10;

The declared part should be conducted within the function, without any initial value.

The defined part should be conducted outside the function, with or without value.

Eg:

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

void fun1()
{
extern int  a; /* declared */
printf("a = %d\n",++a);
}

int  a=10; /* defined */

void fun2()
{
printf("a = %d\n",++a);
}

void main()
{
clrscr();
```

```
fun1();  
fun2();  
printf("a = %d",++a);  
getch();  
}
```

Output:

a = 11

a = 12

a = 13

DIFFERENCES B/W GLOBAL AND EXTERNAL VARIABLES

- Global variables are declared outside the function/top and defined at a time.
- External variables should be declared inside the function and defined outside.
- Global variables doesn't require extern keyword
- Global and external variables available only to the functions declared after the variables.

| Storage class identifier | Storage area | Initial values | Scope | Life time |
|---------------------------------|------------------------------------|-----------------------|------------------------|--------------------------------------|
| Auto | Stack | Garbage | Within function | Until that function executed |
| Register | Cpu registers | Garbage | Within function | Until that function executed |
| Static | Static area of data segment | 0 | Within function | Until total program executed |
| Extern | Data segment | 0 | Total program | Until total program executed. |