

Passing parameters to the functions: [parameter passing techniques]

In C-Language, we can send the arguments to the functions in 2 ways.

1. Call by value / pass by value.
2. Call by address / pass by address. [call by reference]

Call by value / pass by value:

In call by value we are sending actual parameter value to the formal parameter. Later there is no relation is maintained in between actual and formal parameters. Due to this any change in formal parameter doesn't effects the value of actual parameter.

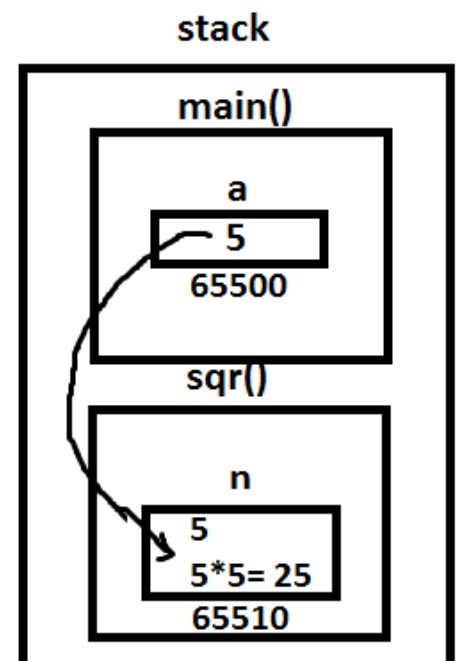
Eg: 1

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

void sqr(int n)
{
    n = n * n;
} /* n deleted after the function execution */

void main()
{
    int a=5;
```

CALL BY VALUE



```
clrscr();  
printf("Before function call a = %d\n",a);  
sqr(a); /* fun calling */  
printf("After function call a = %d", a):  
getch();  
}
```

Output:

Before function call a = 5

After function call a = 5

Eg: 2 swapping of two integers

```
#include<stdio.h>  
#include<conio.h>  
void swap(int a, int b)  
{  
    int temp=a;  
    a=b;  
    b=temp;  
}  
void main()
```

```

{
int a=5, b=7;
clrscr();
printf("Before fun call a=%d, b=%d\n" , a , b);
swap(a, b);
printf("After fun call a=%d, b=%d", a , b);
getch();
}

```

Output:

Before fun call a=5, b=7

After fun call a=5, b=7

The screenshot shows the Turbo C++ IDE with the following code in the editor:

```

Line 12 Col 6 Insert Indent Tab Fill Unindent * E:NONAME.C
#include<stdio.h>
#include<conio.h>
void show(int x) /* fun def, x is formal par */
{
x=200;
} /* x deleted */
void main()
{
int x=100; /* local var */
clrscr();
printf("Before fun call x=%d\n",x);
show(x); /* fun calling , x is actual par */
printf("After fun call x=%d",x);
getch();
}

```

The IDE interface includes a menu bar (File, Edit, Run, Compile, Project, Options, Debug, Break/watch), a toolbar, and a status bar at the bottom showing system information (6:44 PM, 2/13/2023) and a Windows activation notice.

```
Before fun call x=100
After fun call x=100
```

TC

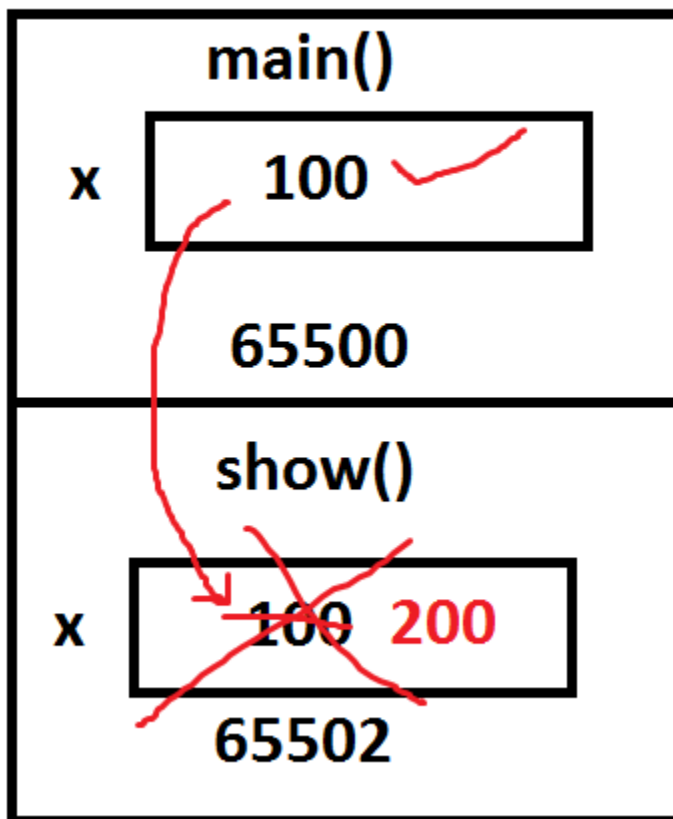
You are screen sharing Stop Share

Mute Start Video Security Participants Chat New Share Pause Share Annotate Apps More

Activate Windows
Go to PC settings to activate Windows.

6:44 PM
2/13/2023

Call by /pass By value



Call by address [Reference]:

In call by address, the address of actual parameter is passed to formal parameter. Due to this the formal parameter should be declared as a pointer. Then only the formal parameter receives the actual parameter address. Due to this any changes in formal parameter effects in actual parameter address i.e. actual parameter value.

Hence pointers allows the local variables to access outside the functions and this process is called call by address / reference.

It is very much useful in handling the strings, arrays etc outside the functions.

Eg: 1

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

```
void sqr(int *n)
```

```
{
```

```
*n = *n * *n;
```

```
}
```

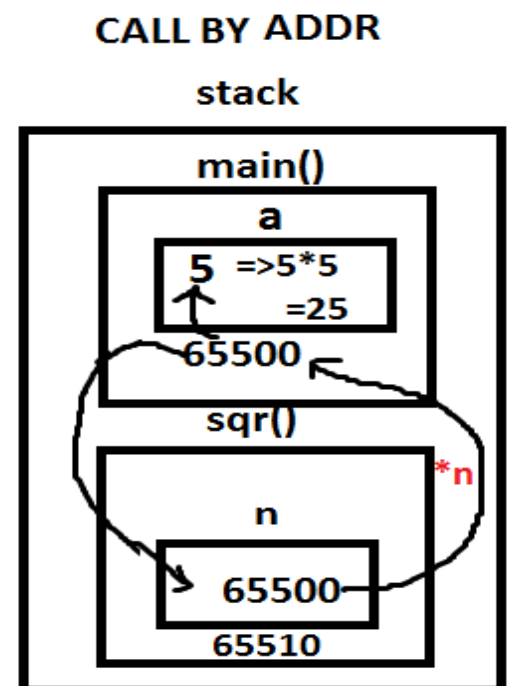
```
void main()
```

```
{
```

```
int a=5;
```

```
clrscr();
```

```
printf("Before function call a = %d\n " ,
```



```
a);  
sqr(&a); /* fun calling with address */  
printf("After function call a = %d " , a);  
getch();  
}
```

Output:

Before function call a = 5

After function call a = 25

Eg: 2 Swap of two integers

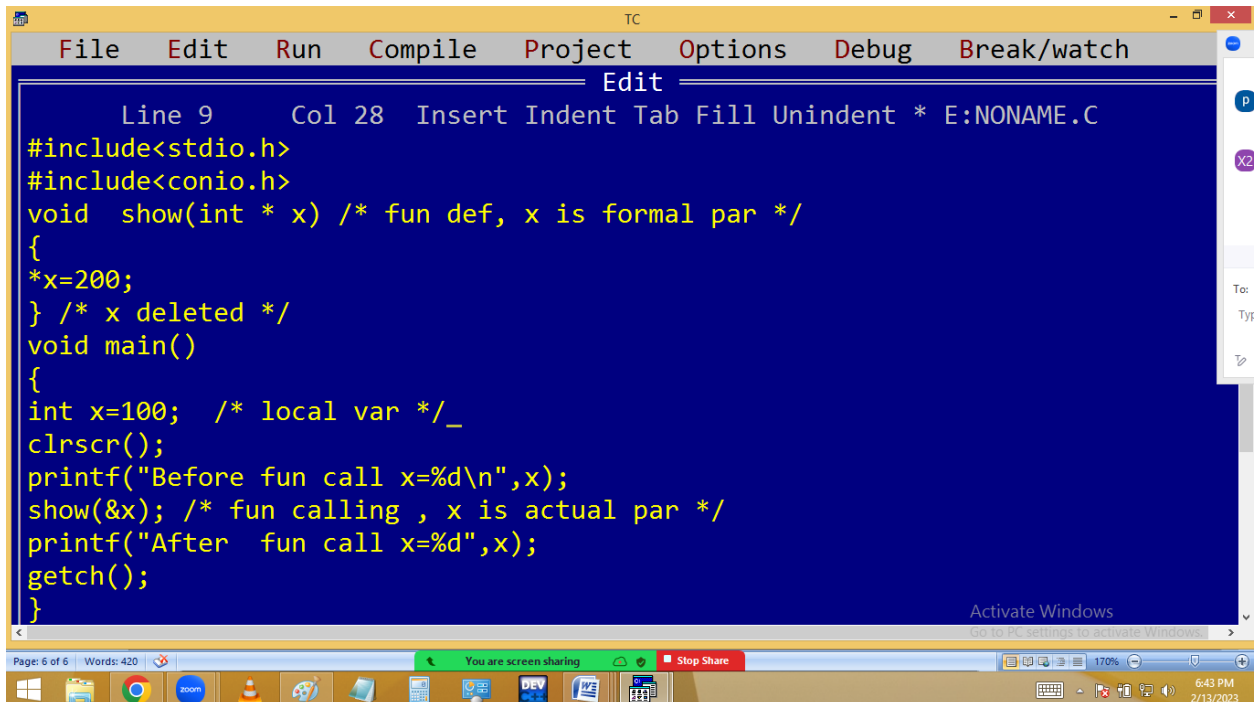
```
#include<stdio.h>  
  
#include<conio.h>  
  
void swap(int *a, int *b)  
{  
    int temp=*a; *a = *b; *b=temp;  
}  
  
void main()  
{  
    int a=5, b=7;  
    clrscr();
```

```
printf("Before fun call a=%d, b=%d\n", a ,b);  
swap(&a, &b);  
printf("After fun call a=%d, b=%d", a ,b);  
getch();  
}
```

Output:

Before function call a=5, b=7

After function call a=7, b=5



```
TC  
File Edit Run Compile Project Options Debug Break/watch  
Edit  
Line 9 Col 28 Insert Indent Tab Fill Unindent * E:NONAME.C  
#include<stdio.h>  
#include<conio.h>  
void show(int * x) /* fun def, x is formal par */  
{  
  *x=200;  
} /* x deleted */  
void main()  
{  
  int x=100; /* local var */_  
  clrscr();  
  printf("Before fun call x=%d\n",x);  
  show(&x); /* fun calling , x is actual par */  
  printf("After fun call x=%d",x);  
  getch();  
}
```

Page: 6 of 6 Words: 420 You are screen sharing Stop Share 170% 6:43 PM 2/13/2023

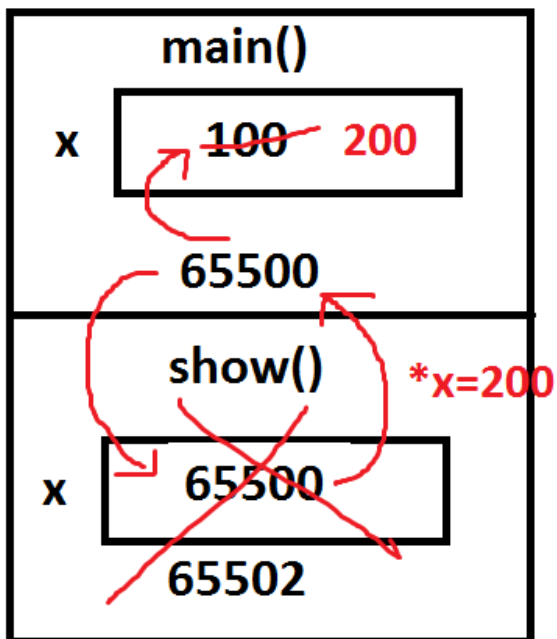
```
TC
Before fun call x=100
After fun call x=200
```

Activate Windows
Go to PC settings to activate Windows.

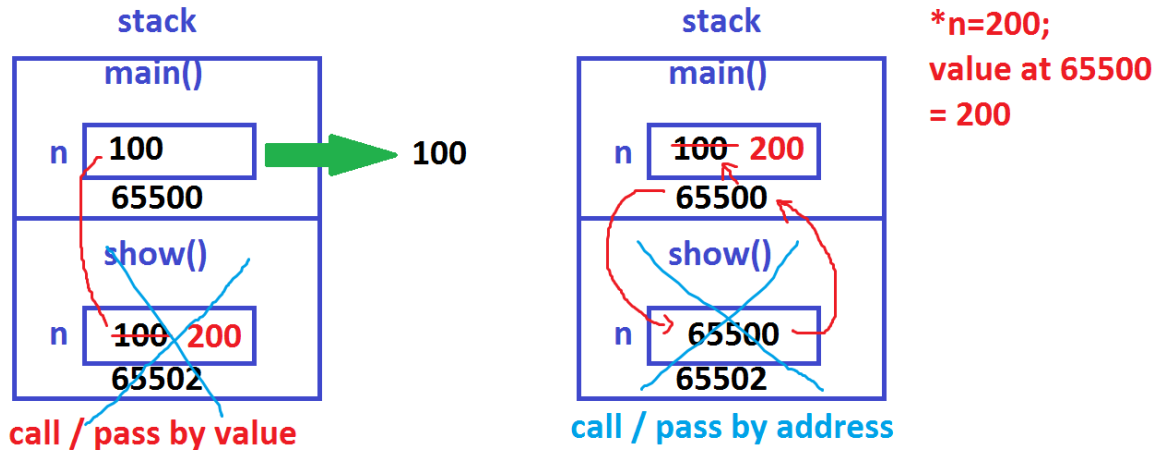
You are screen sharing Stop Share

6:43 PM
2/13/2023

call / pass by address



***x = 200**
x value is 65500
*** means value at 65500 = 200**



PASSING STRING / ARRAY TO FUNCTION

String/array is implicit pointer i.e. string / array variable stores base address. Due to this when string/array is passed to a function, implicitly base address is passed and formal parameter becomes pointer and it receives this address. Hence any change occurred in formal parameter, effects on actual parameter value also.

We can declare string / array formal parameter in 3 ways.

1. With size eg: char st[50] / int a[3]
2. Without size eg: char st[] / int a[]
3. As a pointer eg: char * st / int *a

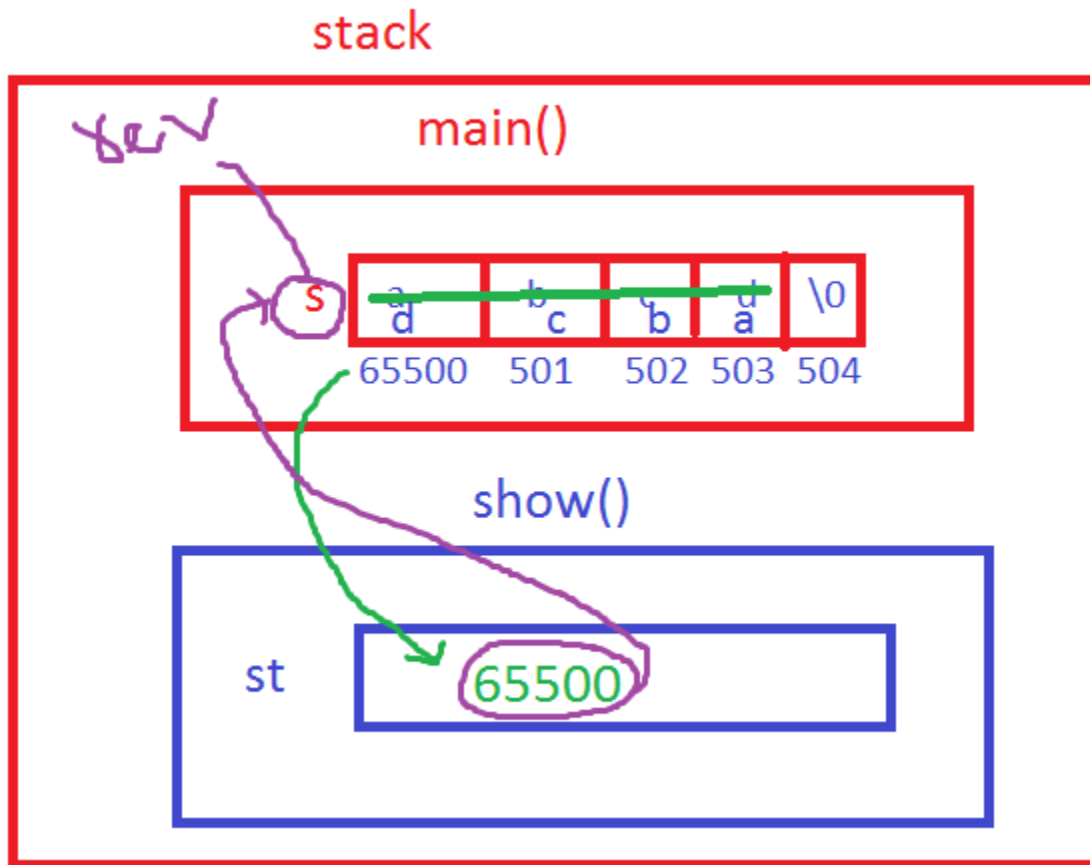
We can pass string / array actual parameter with or without address.

Eg:

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

```
#include<conio.h>
#include<string.h>
void reverse( char st[10] )  or st[ ] or *st
{
    strrev(st);
}
void main()
{
    char s[10]="abcd";
    clrscr();
    reverse(s); or reverse(&s);
    printf("String = %s", s);
    getch();
}
```

O/P: String = dcba



Passing array to function:

#include<stdio.h>

#include<conio.h>

void show(int a[3]) or a[] or *a

```
{  
a[0]=100; a[1]=200; a[2]=300;  
}  
void main()  
{  
int a[3]={10,20,30};  
clrscr();  
show(a); or show(&a);  
printf("Array elements %d %d %d",a[0],a[1],a[2]);  
getch();  
}
```

O/P: Array elements 100 200 300

Passing two – dimensional array to function.

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

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

```
void show( int (*a)[3] ) or a[2][3] or a[ ][3]
```

```
{
```

```
a[0][0]=10; a[1][2]=60;
```

```
}
```

```
void main()
```

```
{
```

```
int a[2][3]={1,2,3,4,5,6};
```

```
show(a); /* fun calling */
```

```
printf("a[0][0]=%d, a[1][2]=%d",a[0][0],a[1][2]);
```

```
getch();
```

```
}
```

Output: a[0][0]=10, a[1][2]=60;

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

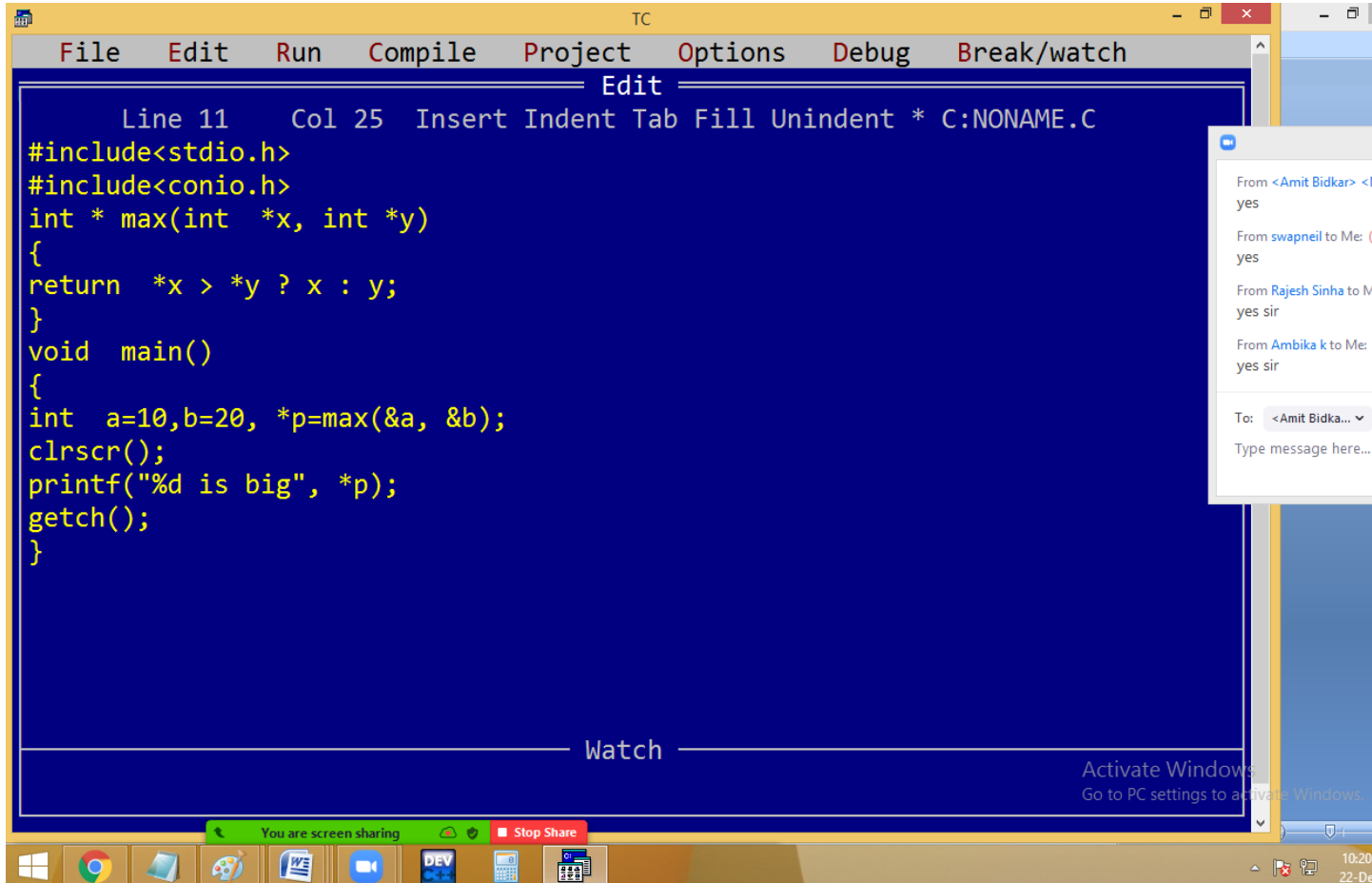
```
#include<conio.h>
```

```
void show(int (*a)[3]) /* or a[2][3] or a[][3]*/
```

```
{
int r,c;
printf("Elements are\n");
for(r=0;r<2;r++)
{
for(c=0;c<3;c++)
{
printf("%4d",*(*(a+r)+c)); /*or a[r][c]*/
}
printf("\n");
}
}

void main()
{
int a[2][3]={1,2,3,4,5,6};
clrscr();
show(a); /* fun calling */
getch();
}
```

Function returning address [pointer]



```
TC
File Edit Run Compile Project Options Debug Break/watch
Edit
Line 11 Col 25 Insert Indent Tab Fill Unindent * C:\NONAME.C
#include<stdio.h>
#include<conio.h>
int * max(int *x, int *y)
{
return *x > *y ? x : y;
}
void main()
{
int a=10,b=20, *p=max(&a, &b);
clrscr();
printf("%d is big", *p);
getch();
}

Watch
Activate Windows
Go to PC settings to activate Windows.

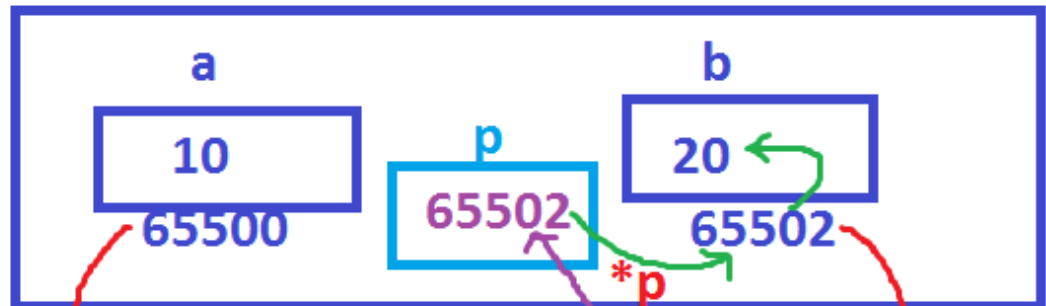
You are screen sharing Stop Share
10:20
22-Dec
```



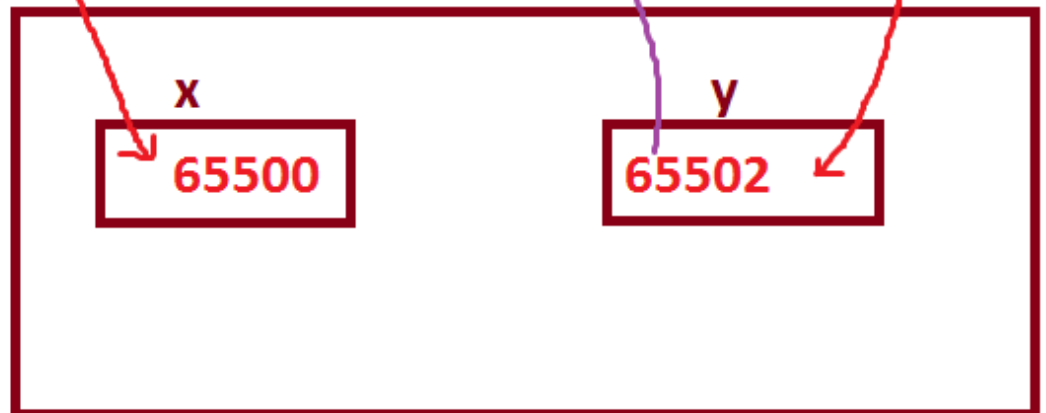
```
20 is big
```

stack

main()



max()



File Edit Run Compile View Help

Mute Start Video Security Participants Chat New Share Pause Share Annotate More

Line 7 Col 1 Insert Indent Tab Fill Unindent * E:REV.C

```
#include<stdio.h>
#include<conio.h>
char * max(int *x, int *y)
{
return *x>*y ? "a is big" : *y > *x ?"b is big":"Both are equal";
}
void main()
{
int a,b; char *p;
clrscr();
printf("Enter a, b values "); scanf("%d %d",&a, &b);
p = max(&a, &b);
printf(p); /* fun calling */
getch();
}
```

Page: 16 of 16 Words: 658

100%

12:26 26-Nov

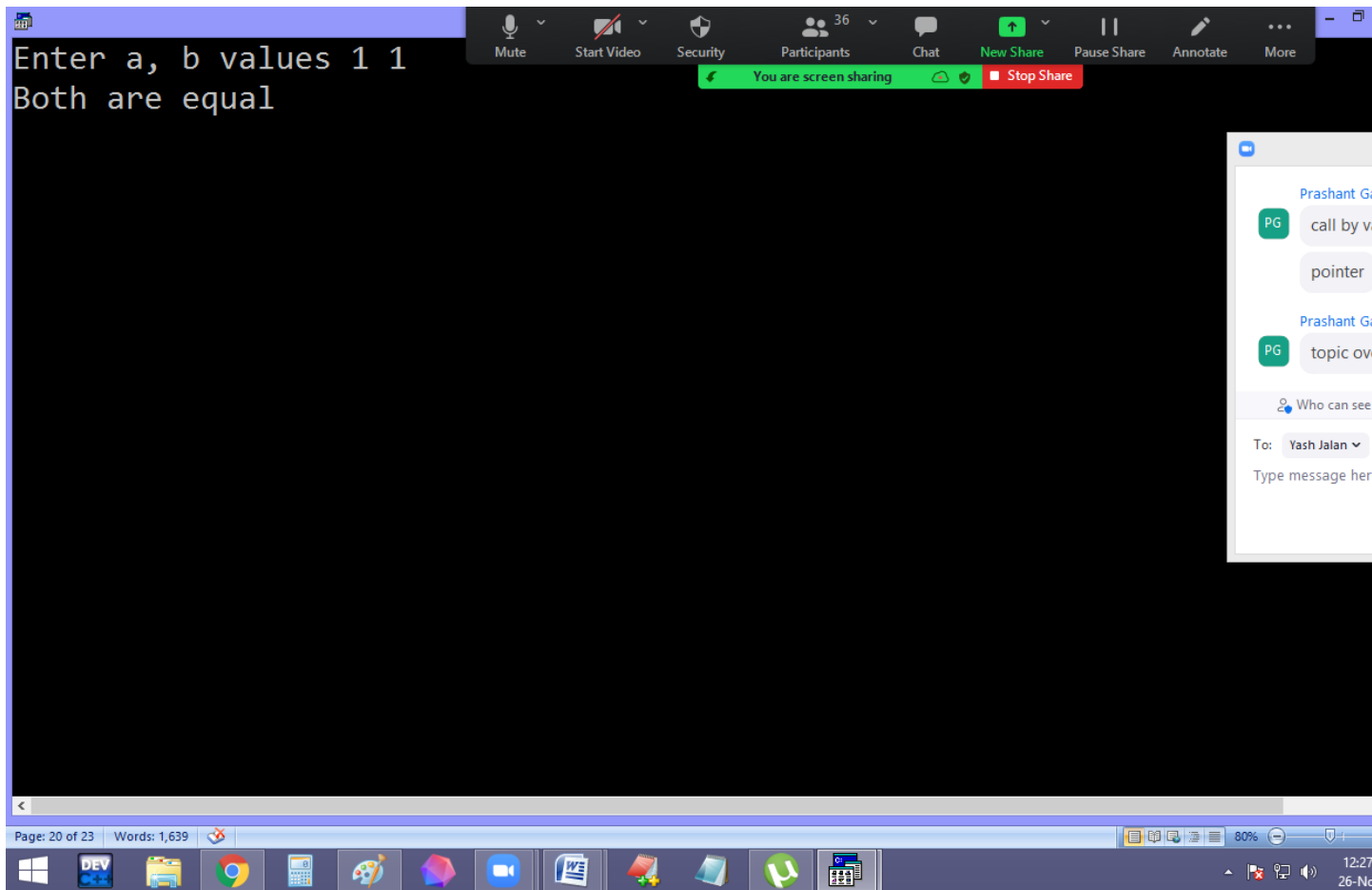
Prashant G
PG call by v
pointer

Prashant G
PG topic ov

Who can see

To: Yash Jalan

Type message here



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.

TC

File Edit Run Compile Project Options Debug Break/watch

Line 7 Col 1 Insert Indent Tab Fill Unindent * E:R1.C

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

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

You are screen sharing Stop Share

6:49 PM 15-Dec-21

TC

```
1
2
3
4
5
6
7
8
9
10
```

You are screen sharing Stop Share

6:49 PM 15-Dec-21

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

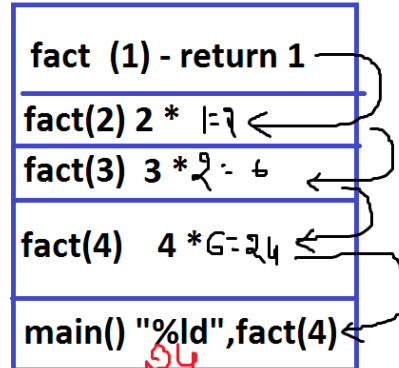
```

long fact(int n)
{
    if(n>1) return n*fact(n-1);
    return 1;
}

void main()
{
    p("4 fct=%ld", fact(4));
}

```

frame/
block

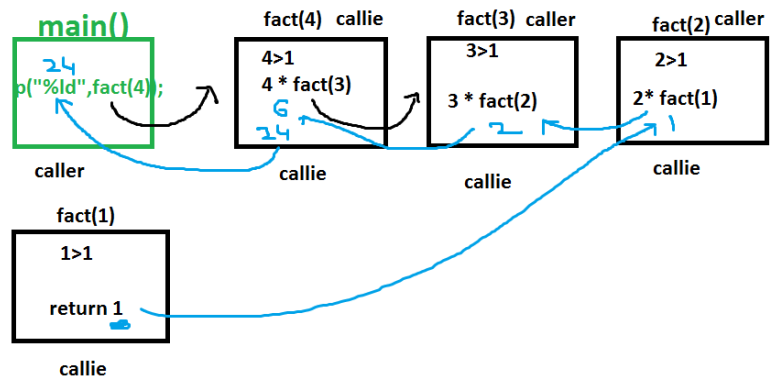


```

long fact(int n)
{
    if(n>1) return n * fact(n-1);
    return 1;
}

void main()
{
    p("4 factorial %ld", fact(4));
    getch();
}

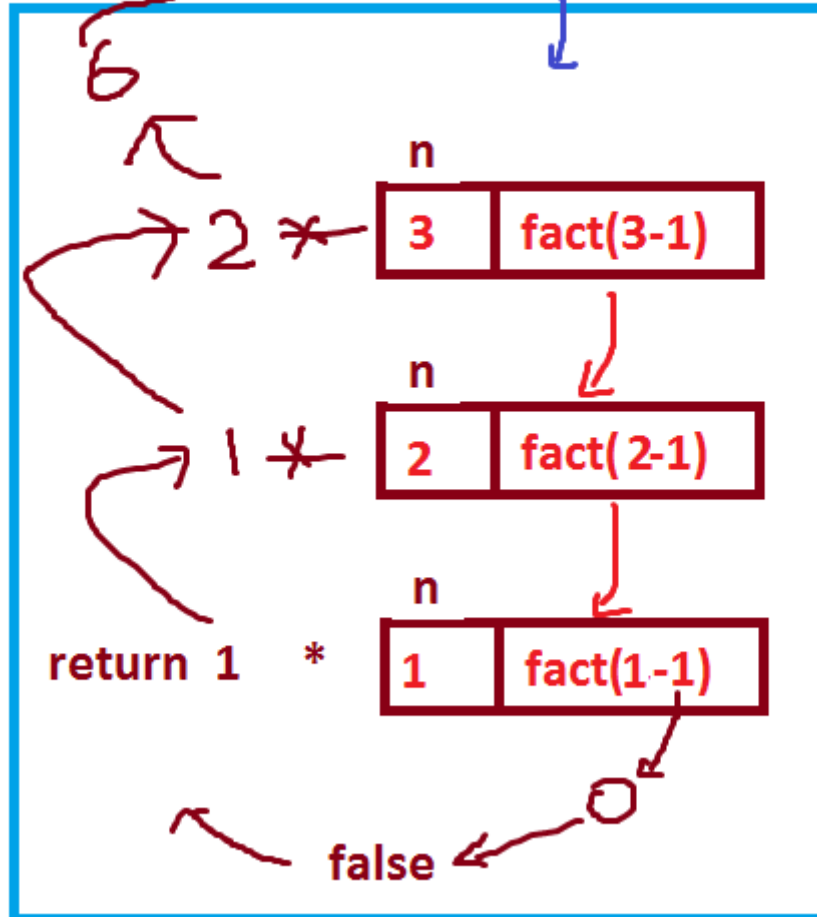
```



main()

{p("%ld",fact(3));}

stack

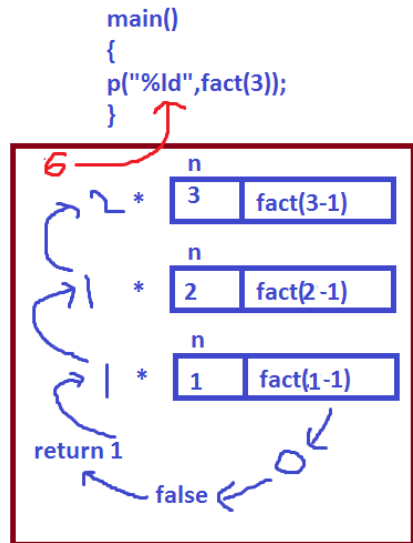



```

#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);

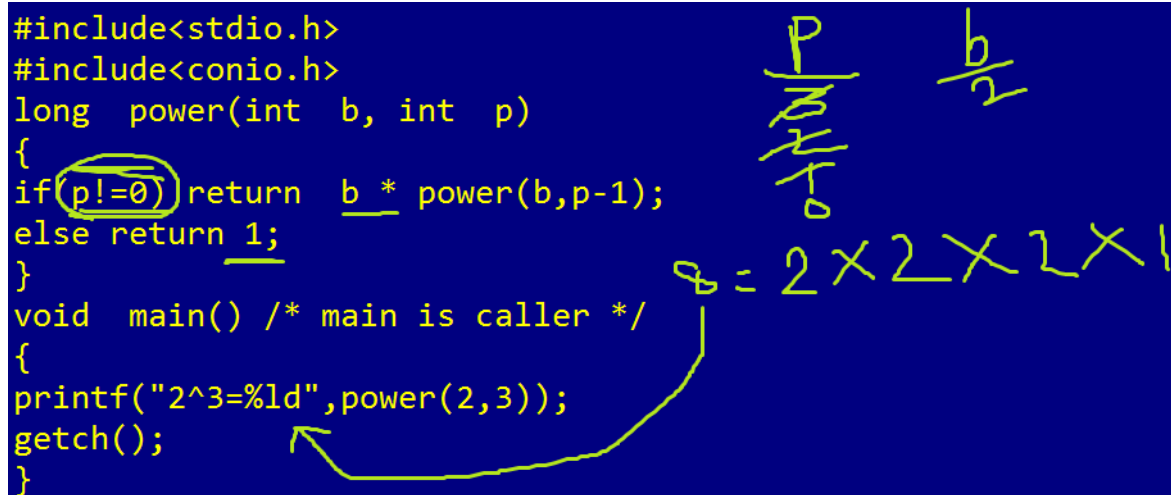
printf("%d ^ %d = %ld", b, p, power(b,p));

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 annotations on the code:

- For the recursive call `power(b,p-1)`, there is a vertical stack of values: $\frac{p}{2}, \frac{2}{1}, \frac{1}{0}$.
- Next to it is a fraction $\frac{b}{2}$.
- Below these, the calculation $2 = 2 \times 2 \times 2 \times 1$ is written.
- An arrow points from the `getch()` line in `main()` to the `2` in the handwritten calculation.

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

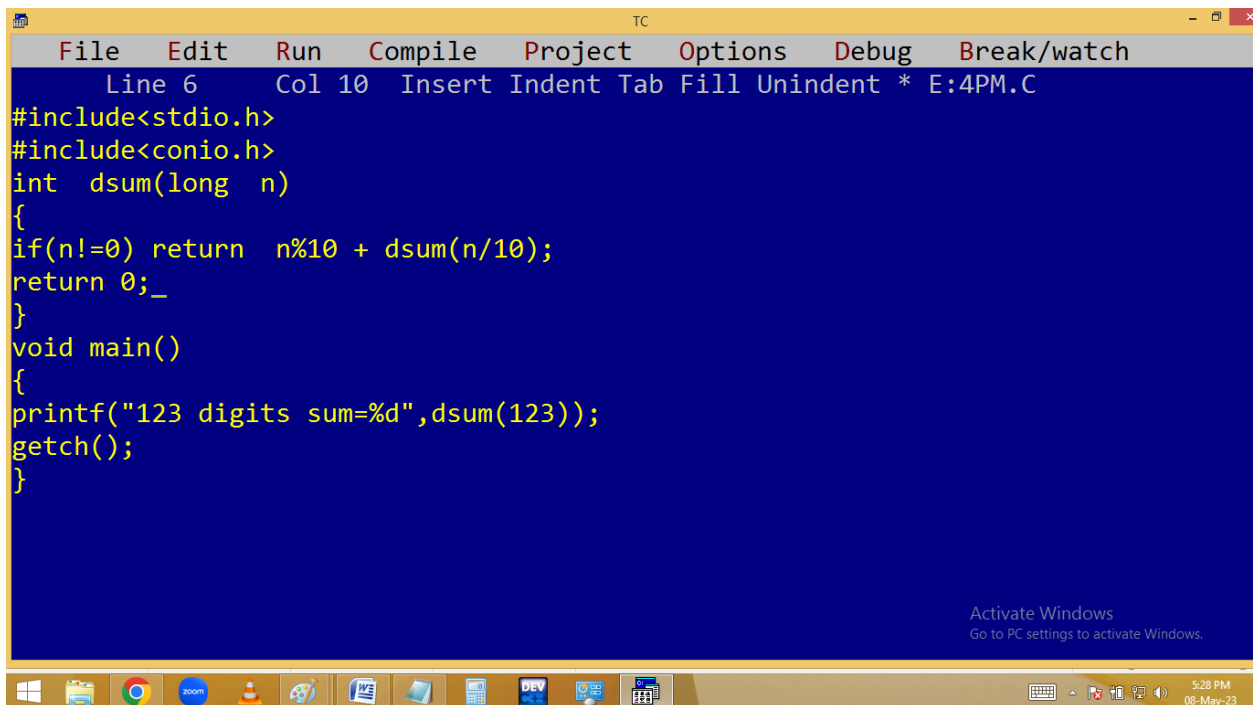
123 digital sum = 6

```
#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();
}
```

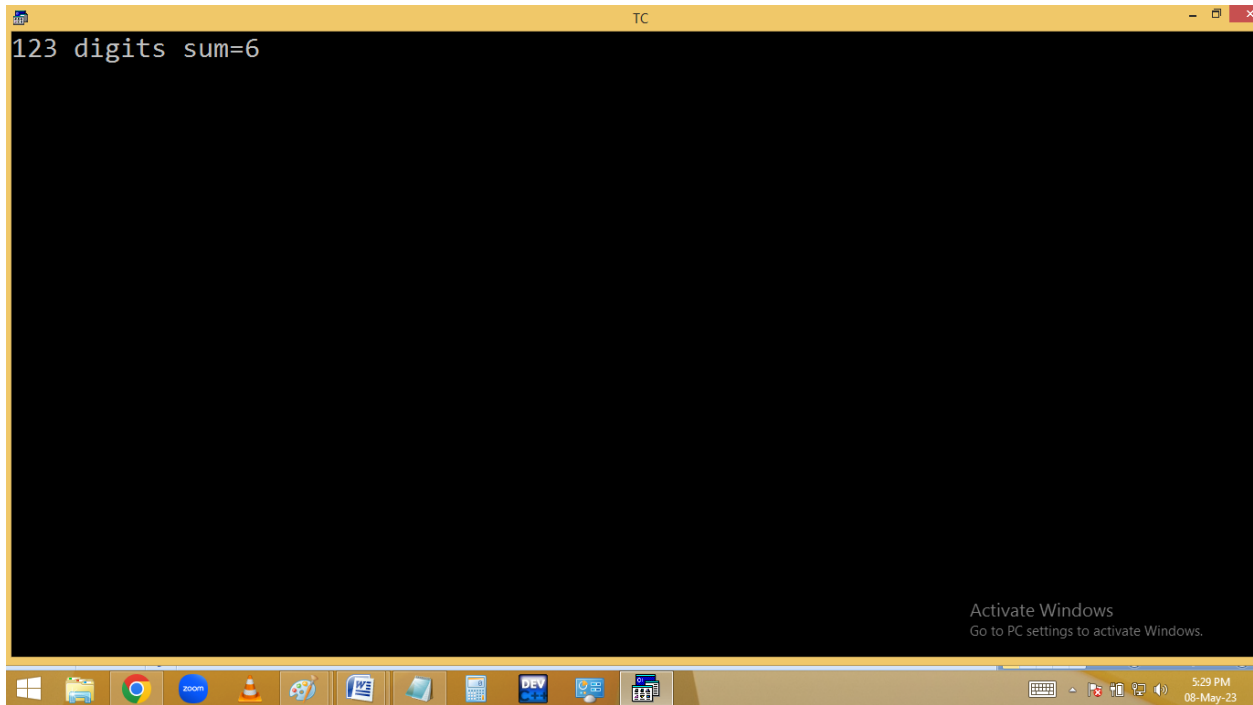
$$\frac{n}{10}$$

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

6



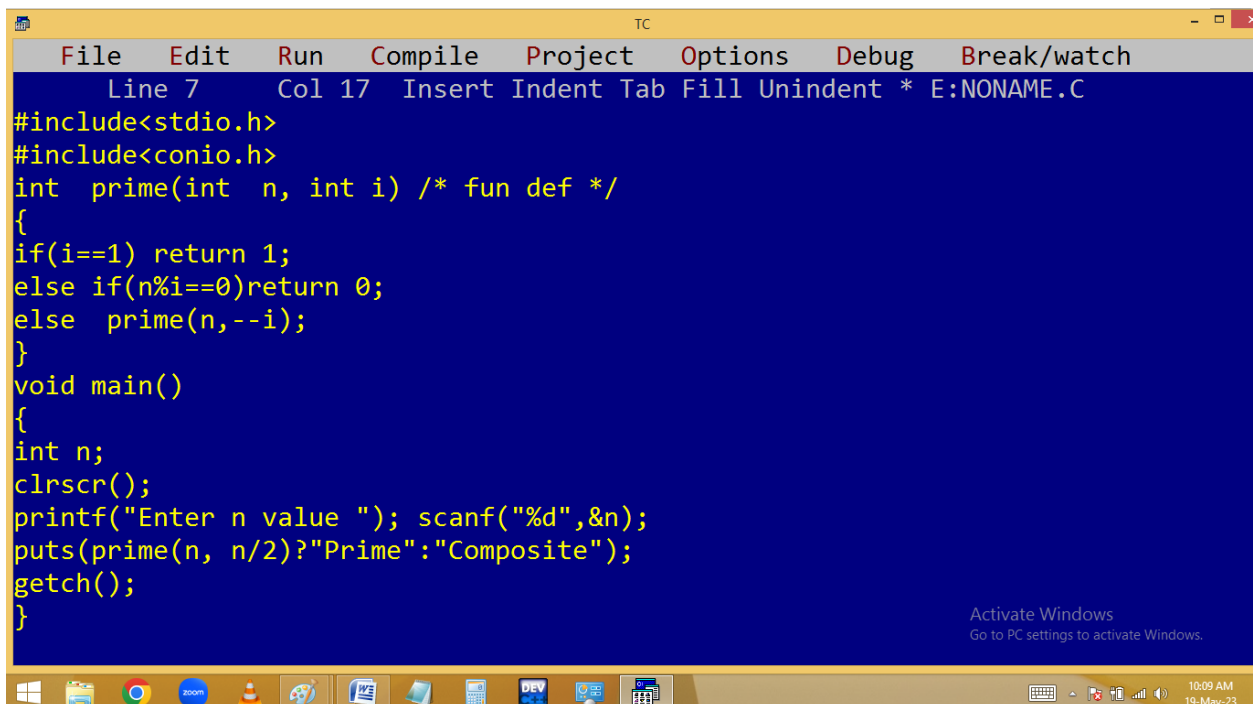
```
TC
File Edit Run Compile Project Options Debug Break/watch
Line 6 Col 10 Insert Indent Tab Fill Unindent * E:4PM.C
#include<stdio.h>
#include<conio.h>
int dsum(long n)
{
    if(n!=0) return n%10 + dsum(n/10);
    return 0;_
}
void main()
{
    printf("123 digits sum=%d",dsum(123));
    getch();
}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



```
123 digits sum=6
```

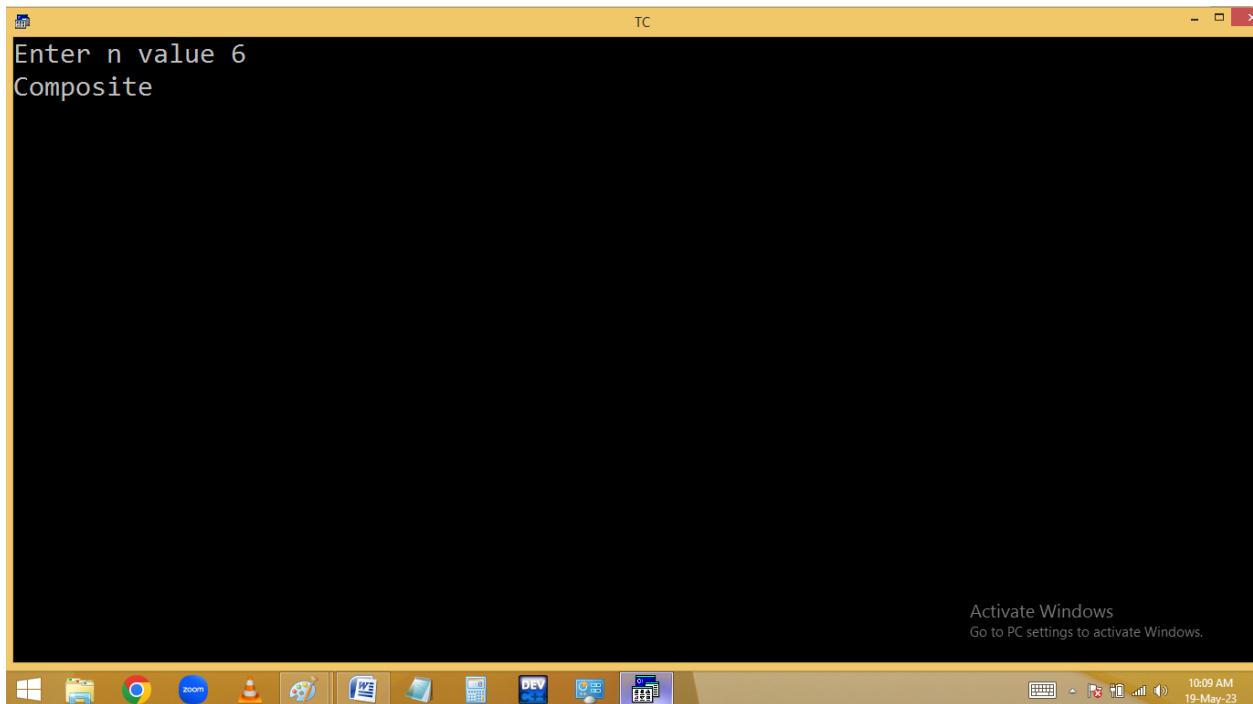
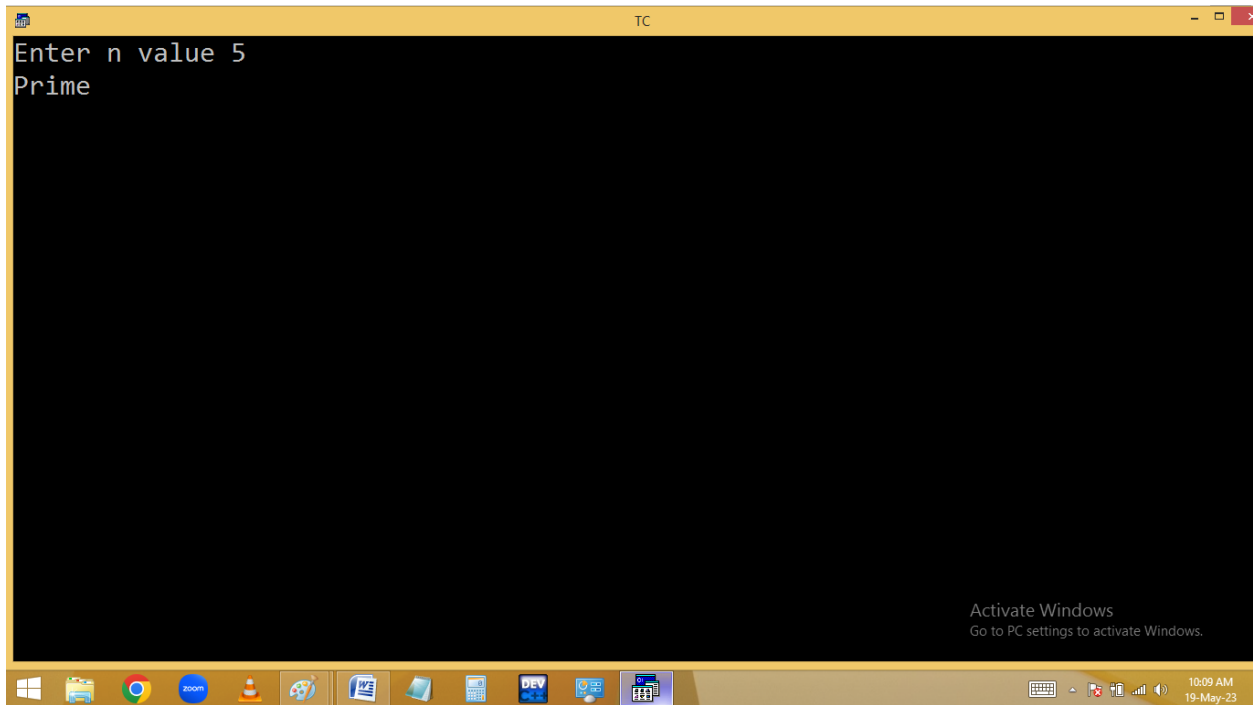
Activate Windows
Go to PC settings to activate Windows.

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();
}
```

Activate Windows
Go to PC settings to activate Windows.



```
TC
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

TC
Activate Windows
Go to PC settings to activate Windows.
```

