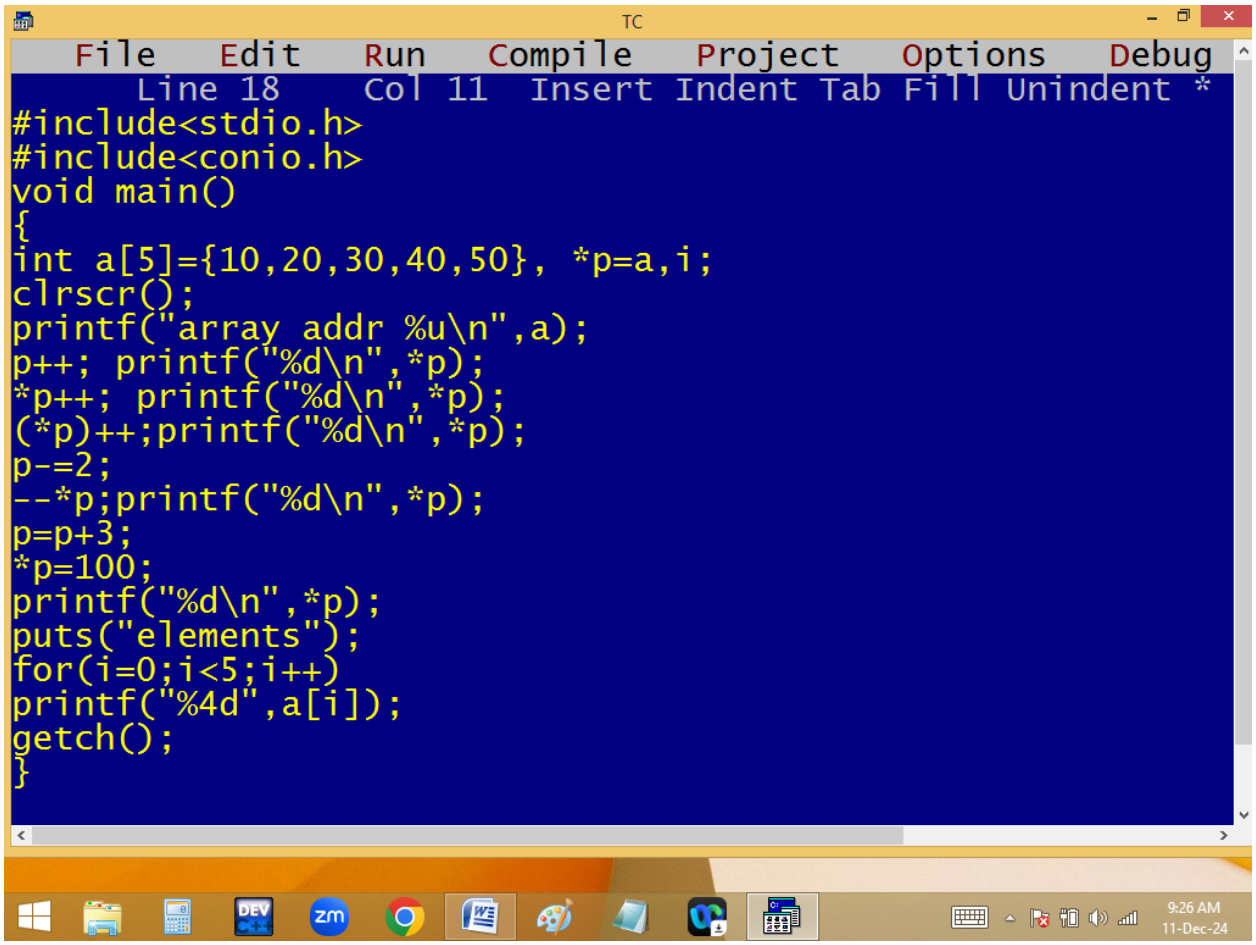
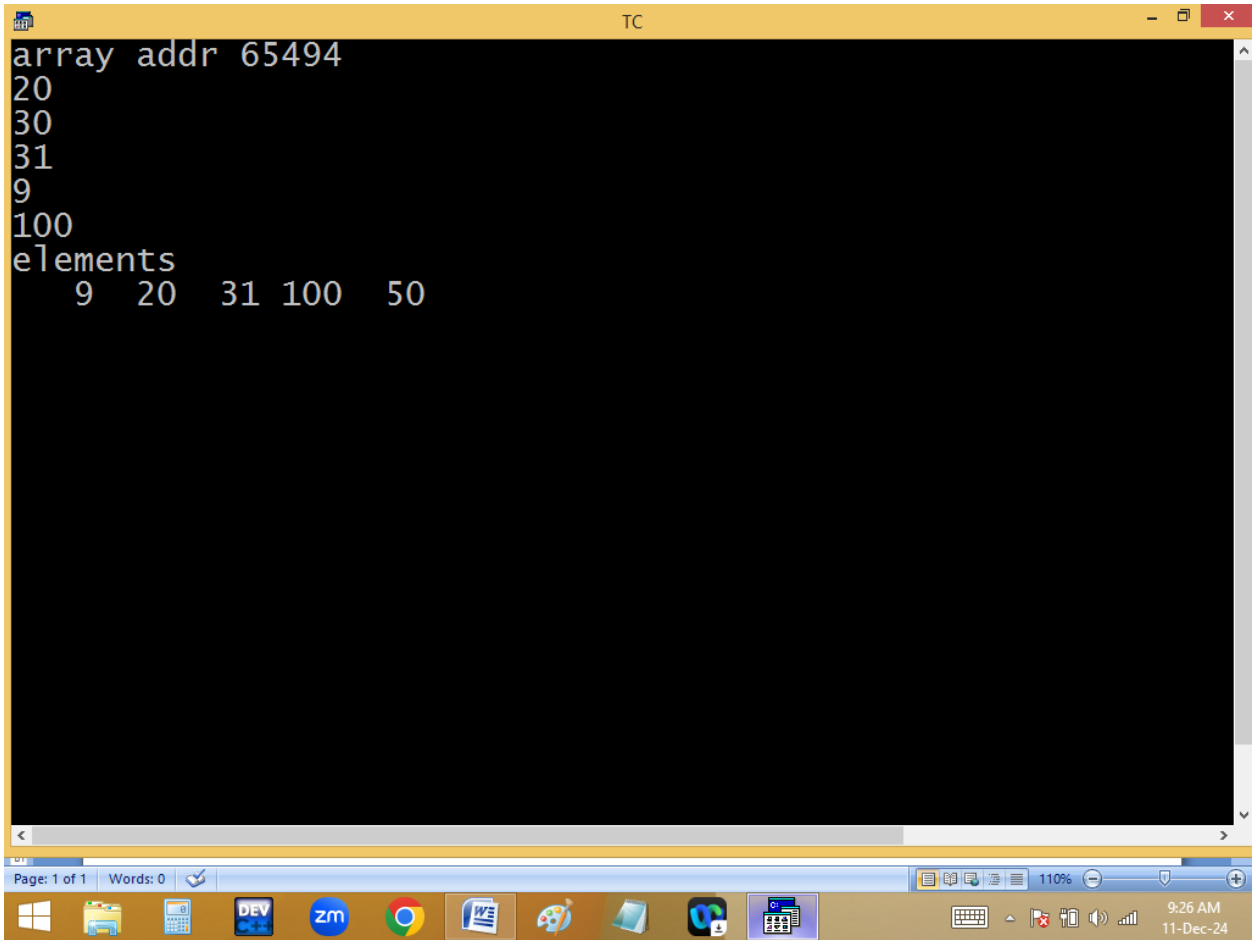

TC
File Edit Run Compile Project Options Debug
Line 18 Col 11 Insert Indent Tab Fill Unindent *
#include<stdio.h>
#include<conio.h>
void main()
{
int a[5]={10,20,30,40,50}, *p=a,i;
clrscr();
printf("array addr %u\n",a);
p++; printf("%d\n",*p);
*p++; printf("%d\n",*p);
(*p)++;printf("%d\n",*p);
p-=2;
--*p;printf("%d\n",*p);
p=p+3;
*p=100;
printf("%d\n",*p);
puts("elements");
for(i=0;i<5;i++)
printf("%4d",a[i]);
getch();
}
9:26 AM
11-Dec-24



p++==>65494+1*2=65496

p(*p)==>value at 65496==>20 ✓

*p++==>65496+1*2=65498

p(*p)==>value at 65498==>30 ✓

(*p)++==>value at 65498++==>30++ ==> 31

p(*p)==>value at 65498 ==> 31 ✓

p=p-2; 65498-2*2=65494

--*p==>value at 65494-- ==> 10-- ==> 9

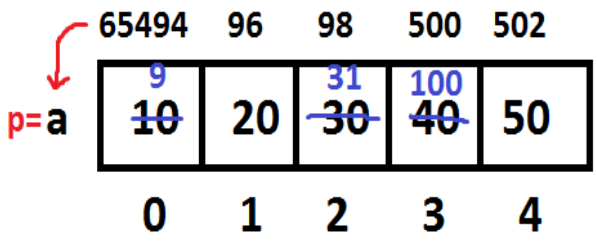
p(*p)==>value at 65494==> 9 ✓

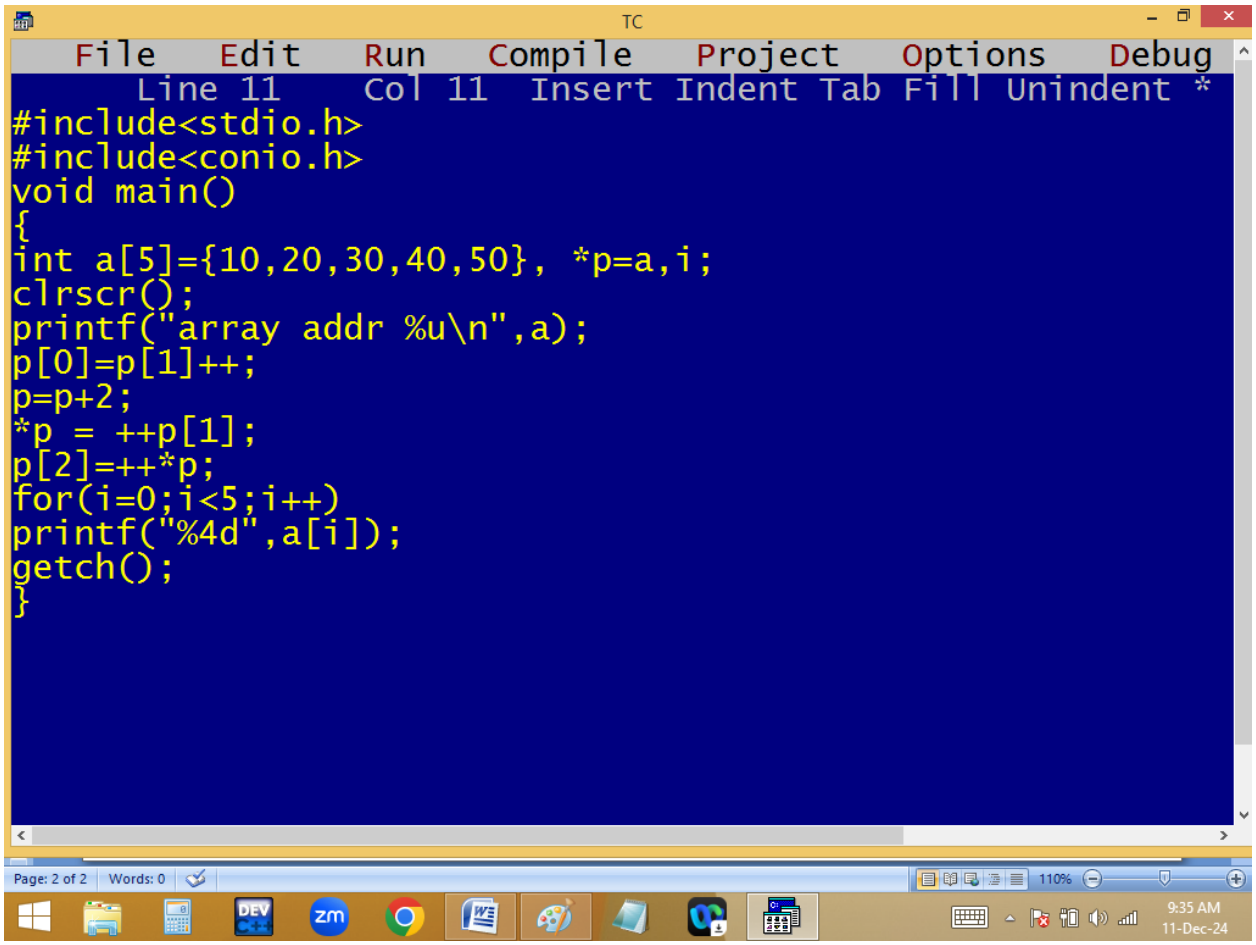
p=p+3;65494+3*2=65500

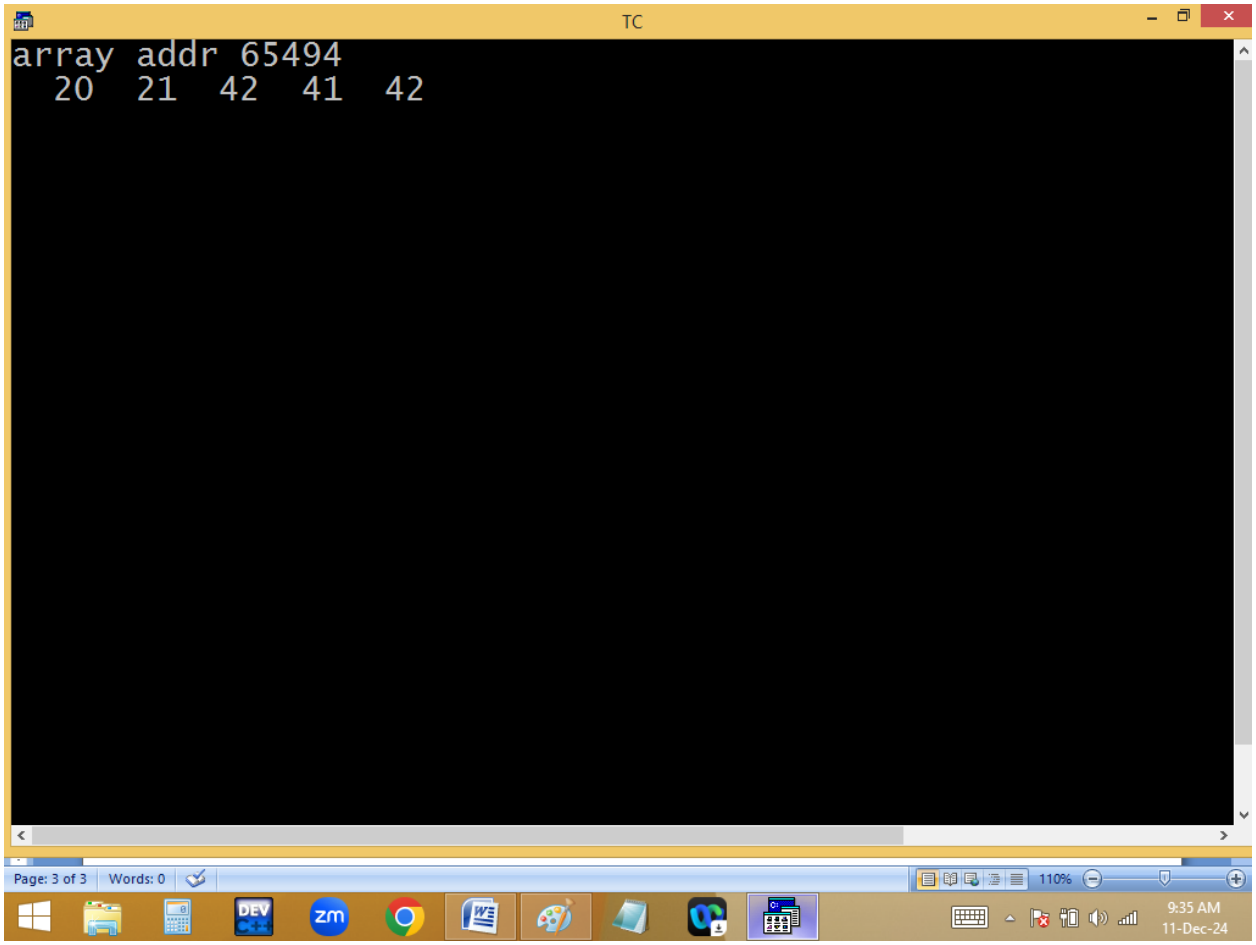
*p=100

value at 65500=100 ==> 40=100

p(*p)==>value at 65500 ==> 100 ✓




TC
File Edit Run Compile Project Options Debug
Line 11 Col 11 Insert Indent Tab Fill Unindent *
#include<stdio.h>
#include<conio.h>
void main()
{
int a[5]={10,20,30,40,50}, *p=a,i;
clrscr();
printf("array addr %u\n",a);
p[0]=p[1]++;
p=p+2;
*p = ++p[1];
p[2]=++*p;
for(i=0;i<5;i++)
printf("%4d",a[i]);
getch();
}
Page: 2 of 2
Words: 0
110%
9:35 AM
11-Dec-24



p[0] = p[1]++; ⟨ p[0]=p[1] / p[1]++

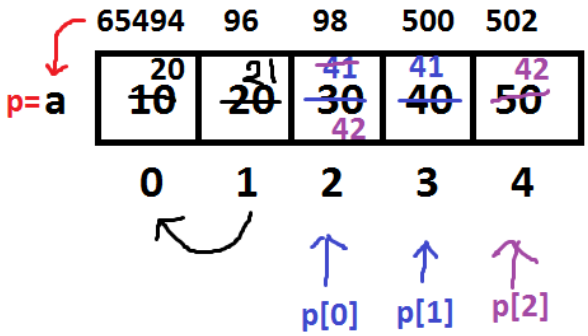
65494 = 65496 ==> 10=20

p[1]++ ==> 20++ ==> 21

p=p+2 ==>65494+2*2=65498

*p = ++p[1]; ⟨ ++p[1] ==> 40++ ==> 41 / *p = p[1] ==> p[0]=p[1]==>30=41

p[2]=++*p; ⟨ ++*p ==> 41++ ==> 42 / p[2]=42



MEMORY MANAGEMENT

To store anything in our computer, we should have to allocate the memory first.

This memory allocation is conducted in two ways.

1. Static memory allocation.
2. Dynamic memory allocation.

In static memory allocation, the memory specified at compile/design time, based on the data type or array size. This type of memory management is called compile time memory management [**compiler indicates memory and O.S allocates the memory**].

In static memory allocation, the memory size is fixed at compile time and we can't

change this memory size at run time. It causes some times memory wastage / shortage.

To avoid this problem, the only solution is dynamic memory allocation.

In dynamic memory allocation, the memory is allocated at run time, based on the user input, instantly.
This type of memory management is called run time memory management.

To conduct dynamic memory allocation, we should have to use **pointers**.

In dynamic memory allocation the memory is allocated in **HEAP** area.

To manage the dynamic memory, we are using some predefined functions like

- malloc()
- calloc()
- realloc()
- free()

All these functions are available in **<alloc.h>**

malloc(), realloc(), calloc() functions are able to allocate the memory of **64KB** Maximum at a time.

To allocate more than 64KB memory, use the functions

- farmalloc()
- farcalloc()
- farrealloc().

Note:

when we are working with dynamic memory allocation, we have to allocate the

memory for any data type. Due to this all these functions return datatype is **void ***, which is a generic type. Due to this we should have to provide **explicit type casting** for all these functions.

malloc()	calloc()
Block Memory allocation	Contiguous memory blocks allocation
Allocates memory in bytes form	Allocates memory in blocks form.
Initial values garbage	Initial values 0
One argument required	Two arguments required
Used for normal variables	Used for array type variables

Syntax:

void * malloc(bytes);

void * calloc(no of blocks, block_size);

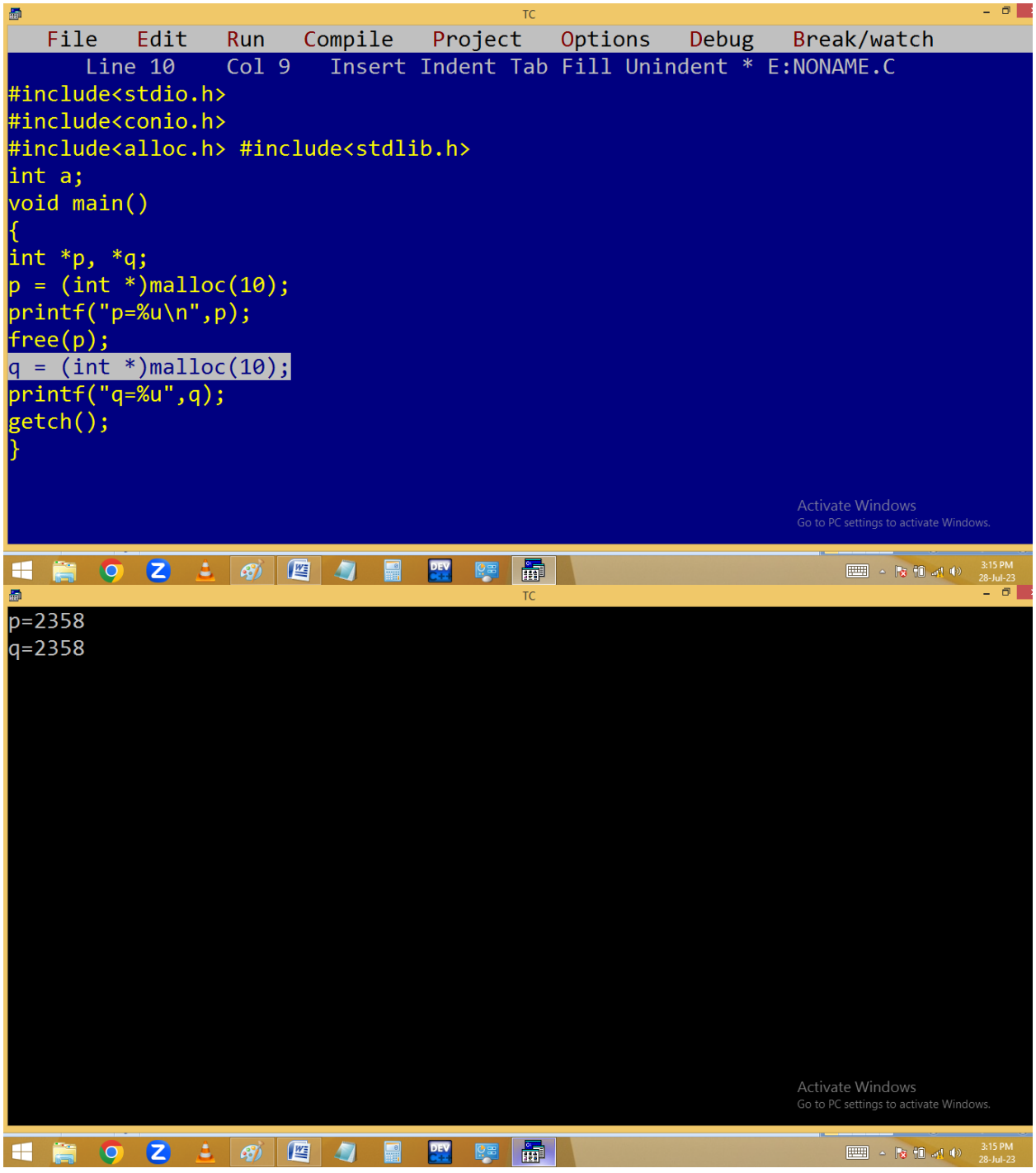
free(): It is used to release the memory allocated by malloc(), calloc() and realloc().

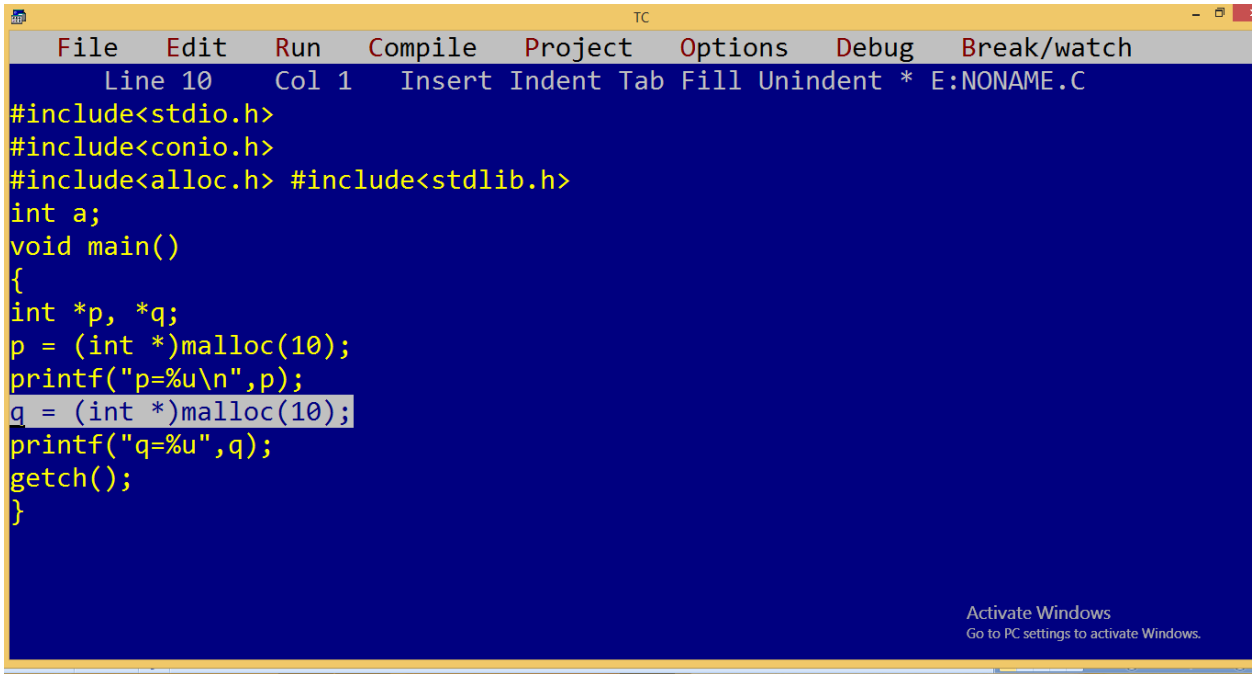
Syntax: void free(pointer);

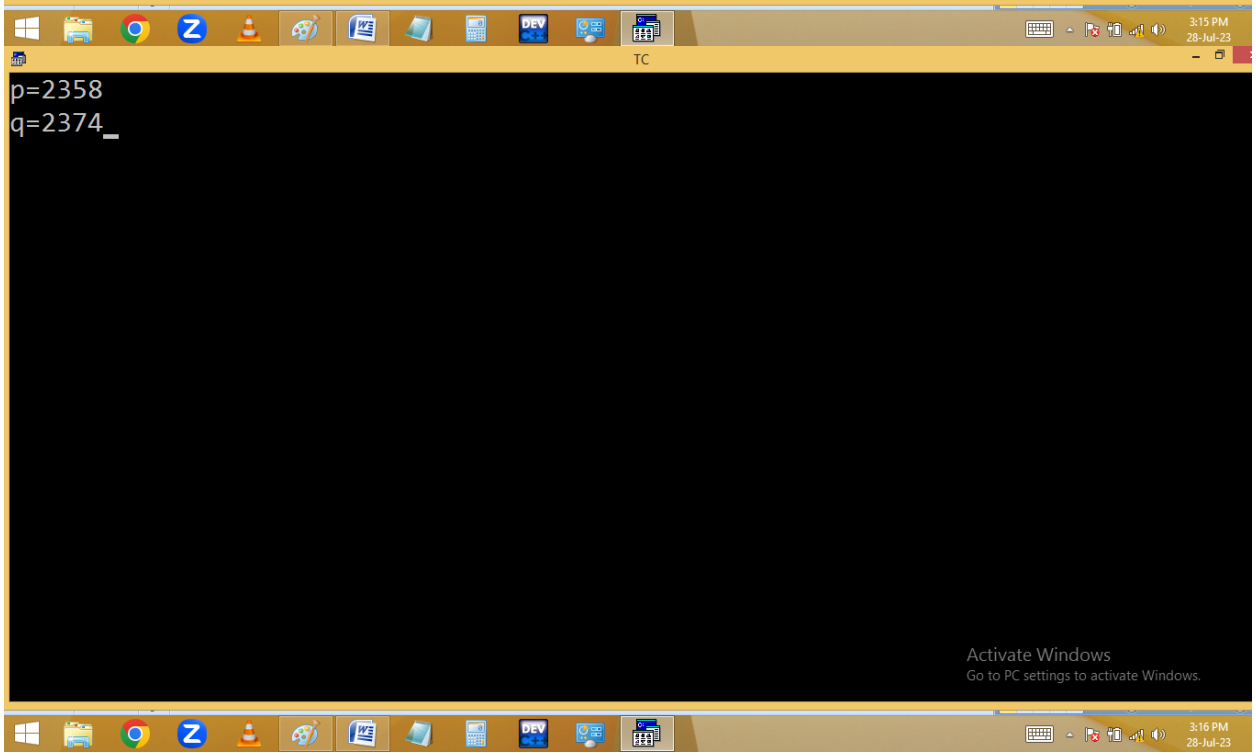
realloc(): It is used to extend the memory allocated by malloc() or calloc() at runtime. Working style is similar to malloc().

Syntax: **void * realloc(oldptr, newsize);**

free example:

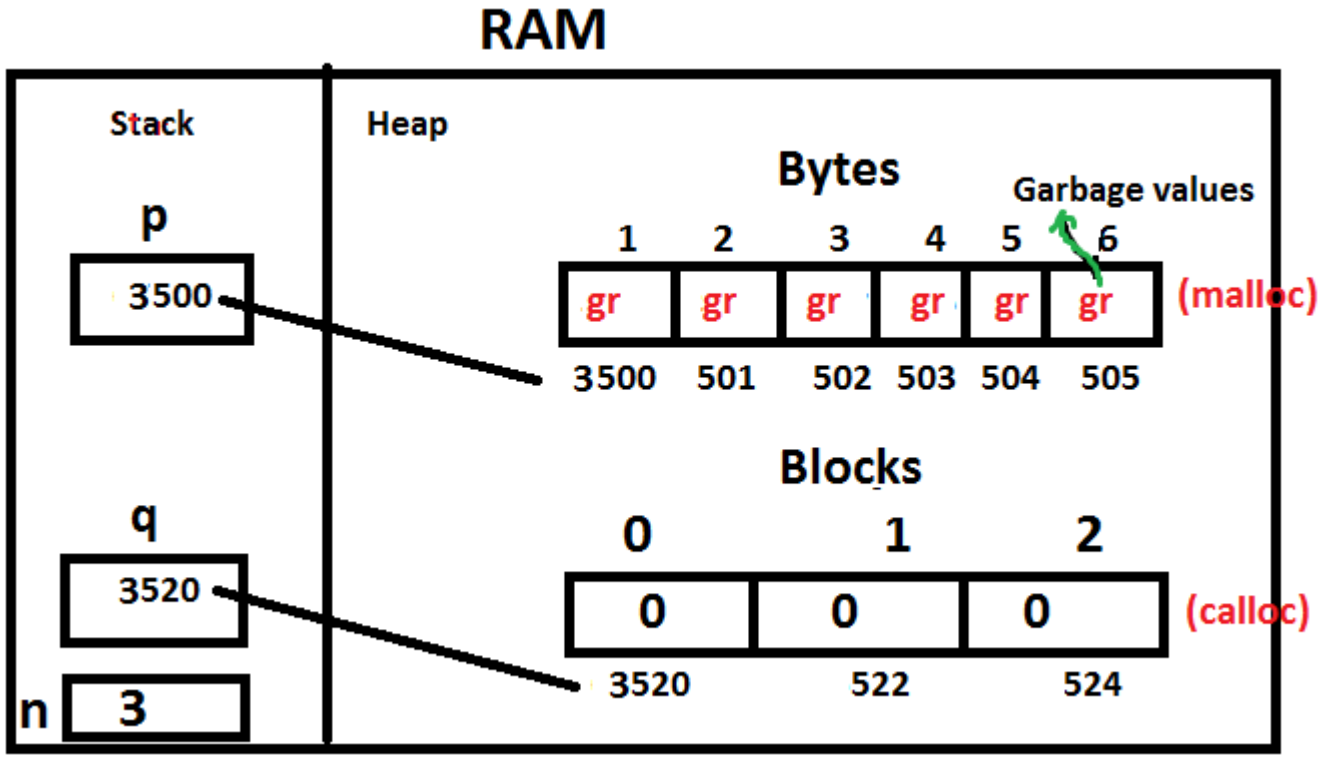

TC
File Edit Run Compile Project Options Debug Break/watch
Line 10 Col 9 Insert Indent Tab Fill Unindent * E:NONAME.C
#include<stdio.h>
#include<conio.h>
#include<alloc.h> #include<stdlib.h>
int a;
void main()
{
int *p, *q;
p = (int *)malloc(10);
printf("p=%u\n",p);
free(p);
q = (int *)malloc(10);
printf("q=%u",q);
getch();
}
Activate Windows
Go to PC settings to activate Windows.
TC
p=2358
q=2358
Activate Windows
Go to PC settings to activate Windows.


TC
File Edit Run Compile Project Options Debug Break/watch
Line 10 Col 1 Insert Indent Tab Fill Unindent * E:NONAME.C
#include<stdio.h>
#include<conio.h>
#include<alloc.h> #include<stdlib.h>
int a;
void main()
{
int *p, *q;
p = (int *)malloc(10);
printf("p=%u\n",p);
q = (int *)malloc(10);
printf("q=%u",q);
getch();
}
Activate Windows
Go to PC settings to activate Windows.

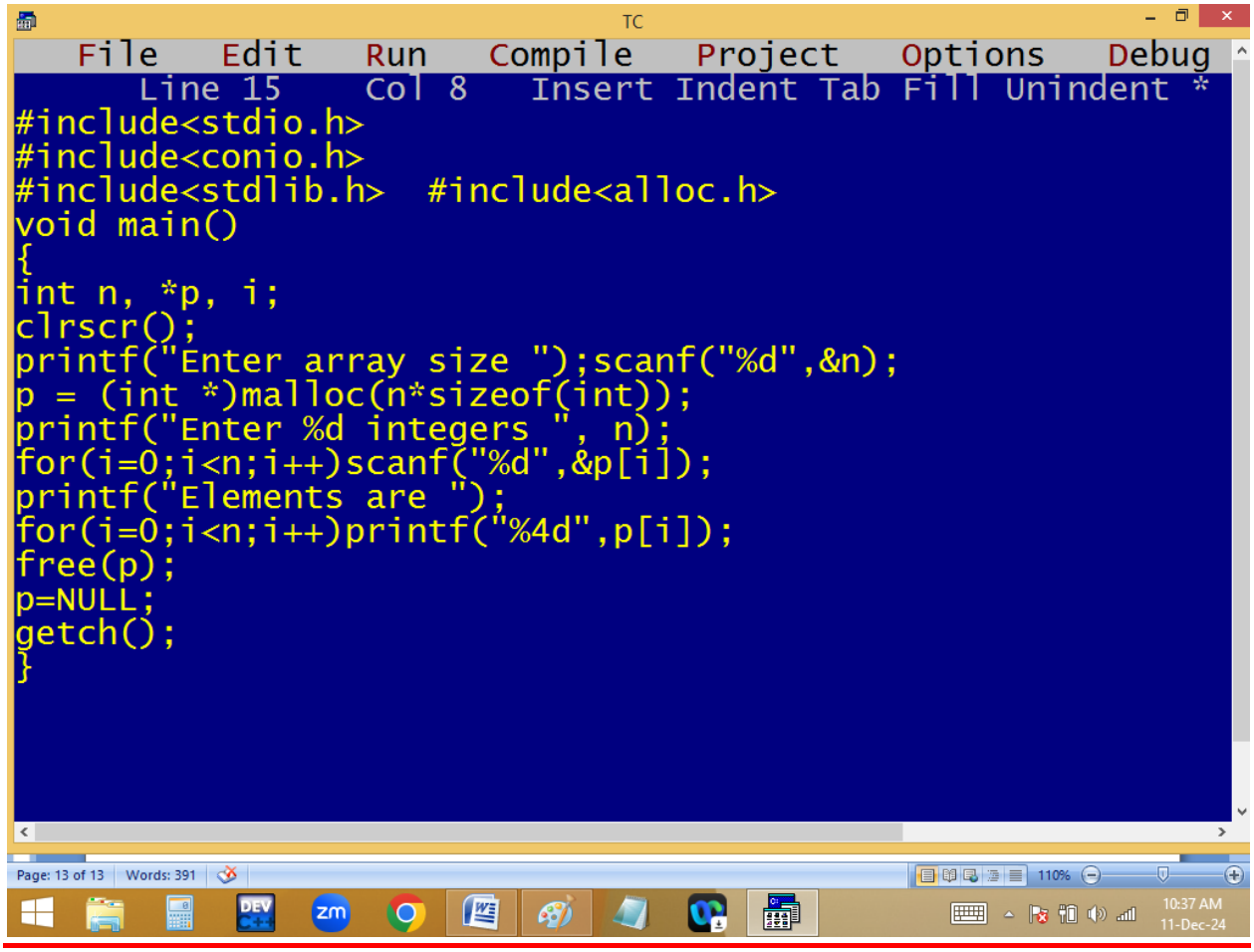

TC
p=2358
q=2374
Activate Windows
Go to PC settings to activate Windows.

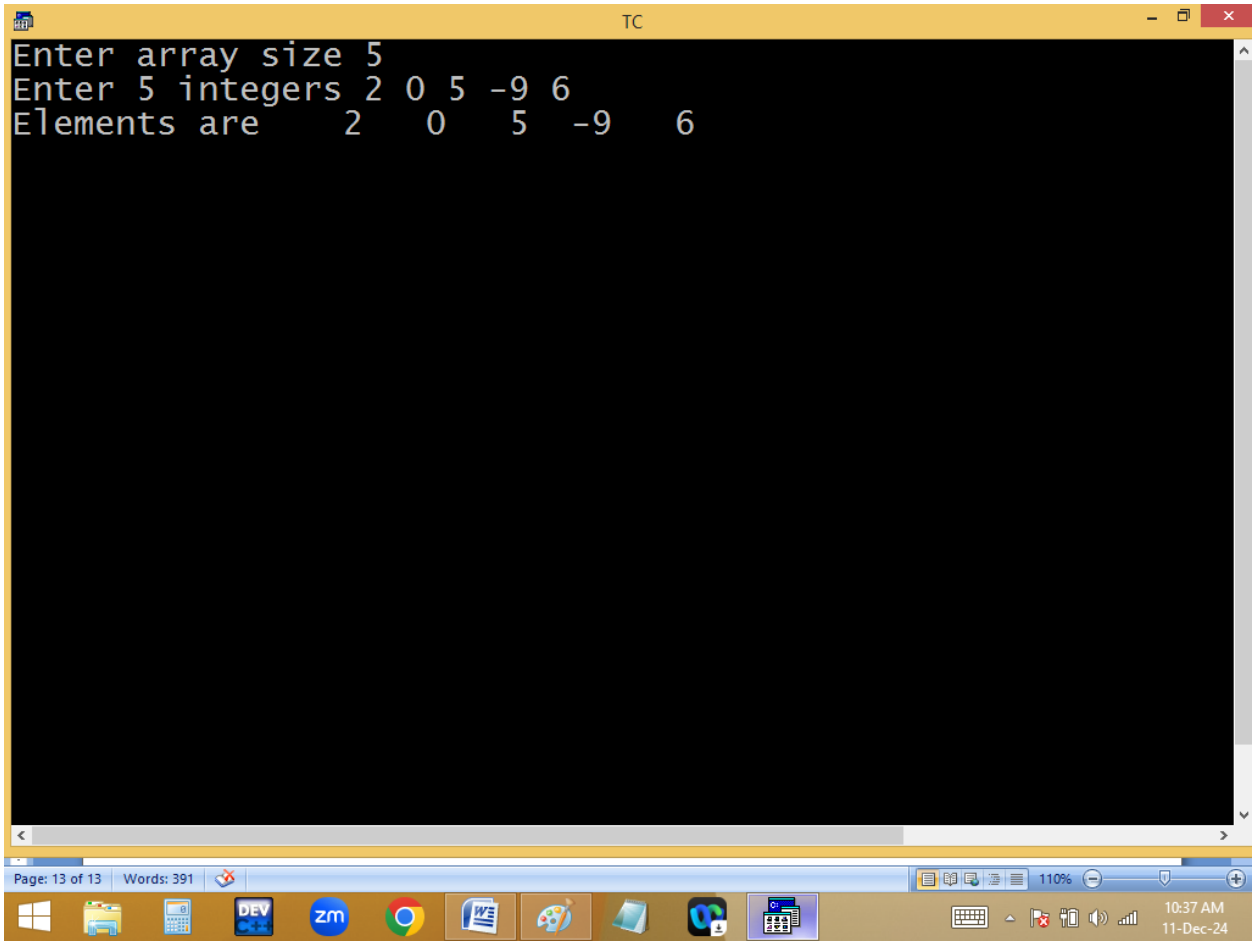
allocating memory for 3 integers using malloc(), calloc().

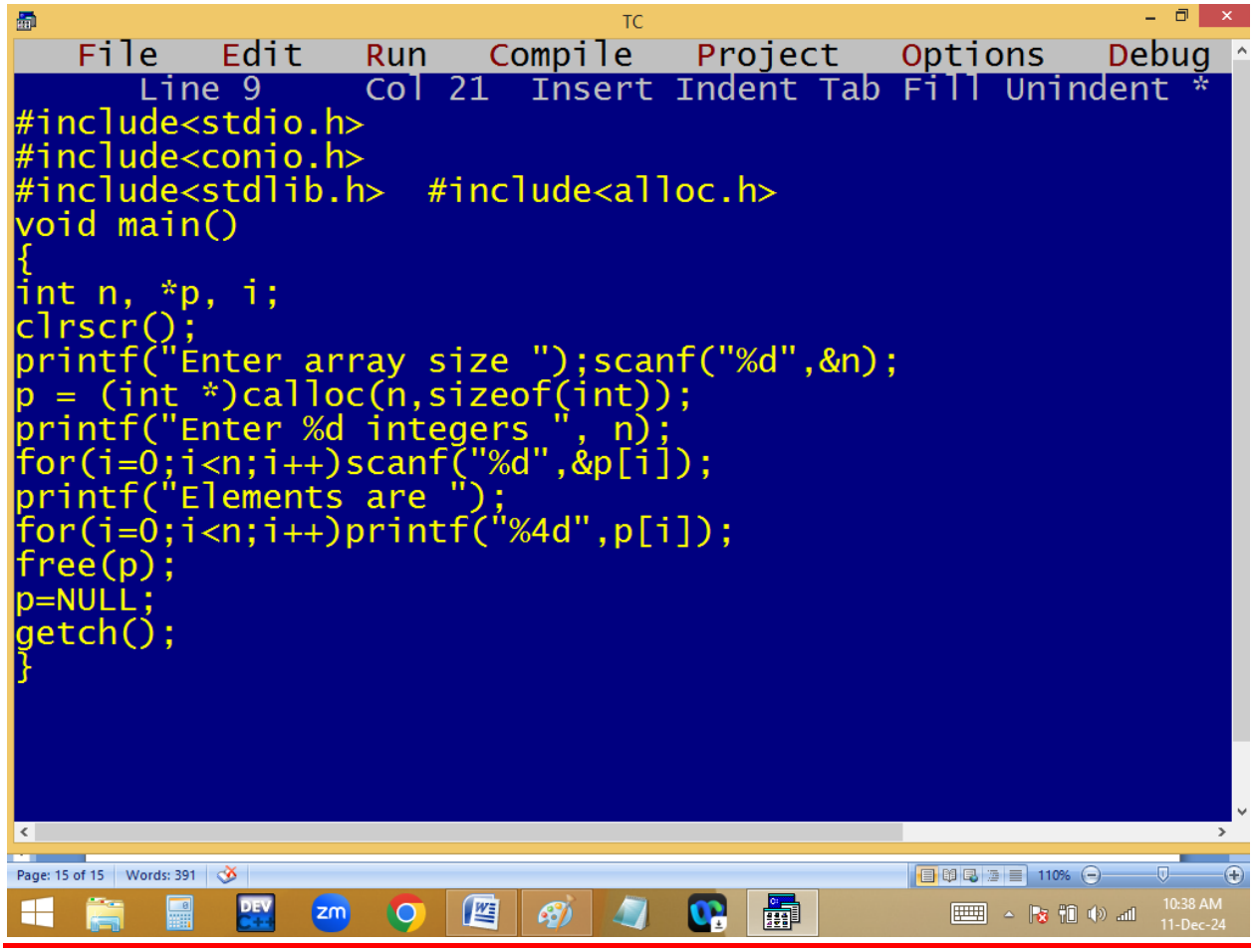
```
int *p, *q, n=3;

p = (int  *)malloc(n * sizeof(int));

q = (int  *)calloc(n ,  sizeof(int));
```

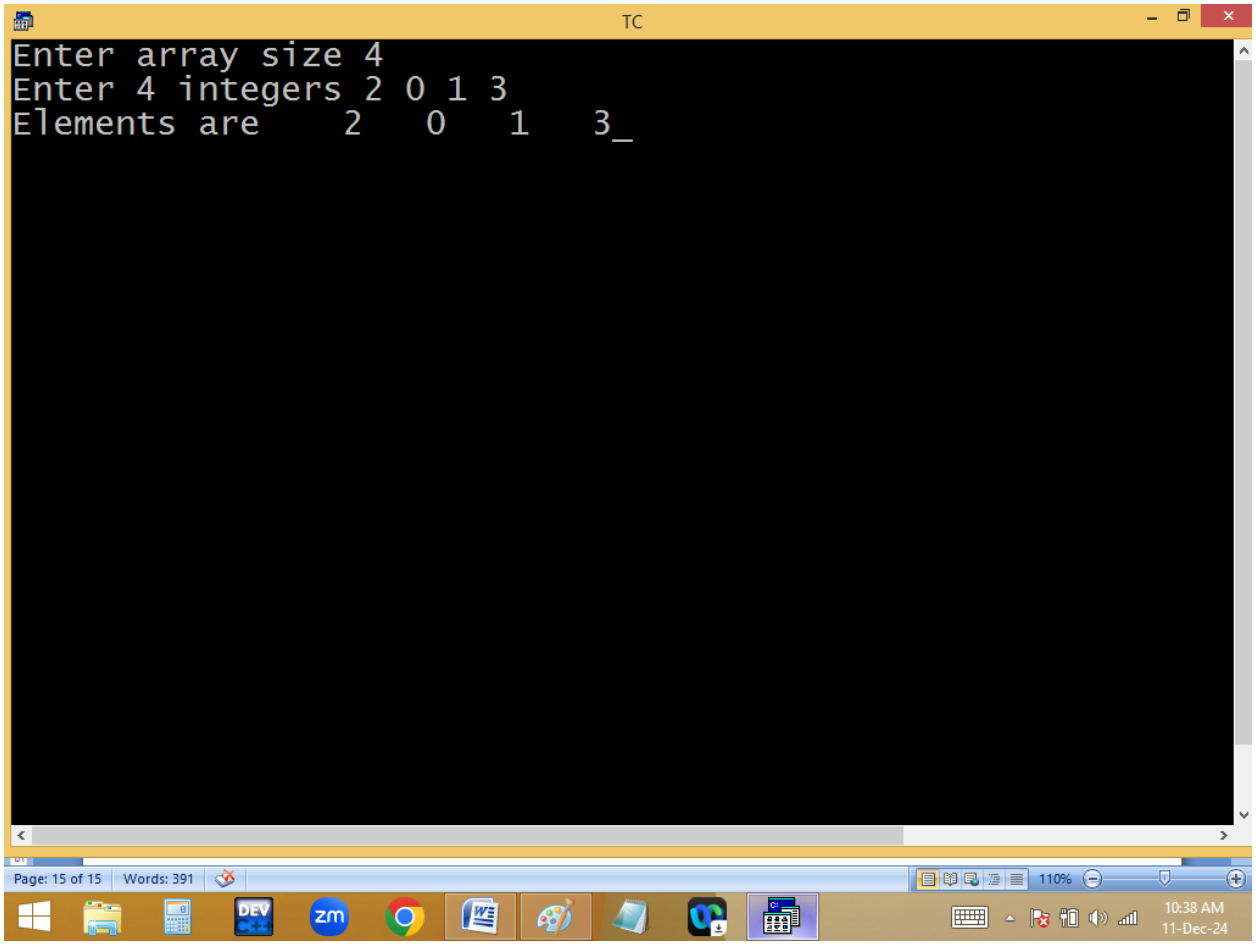


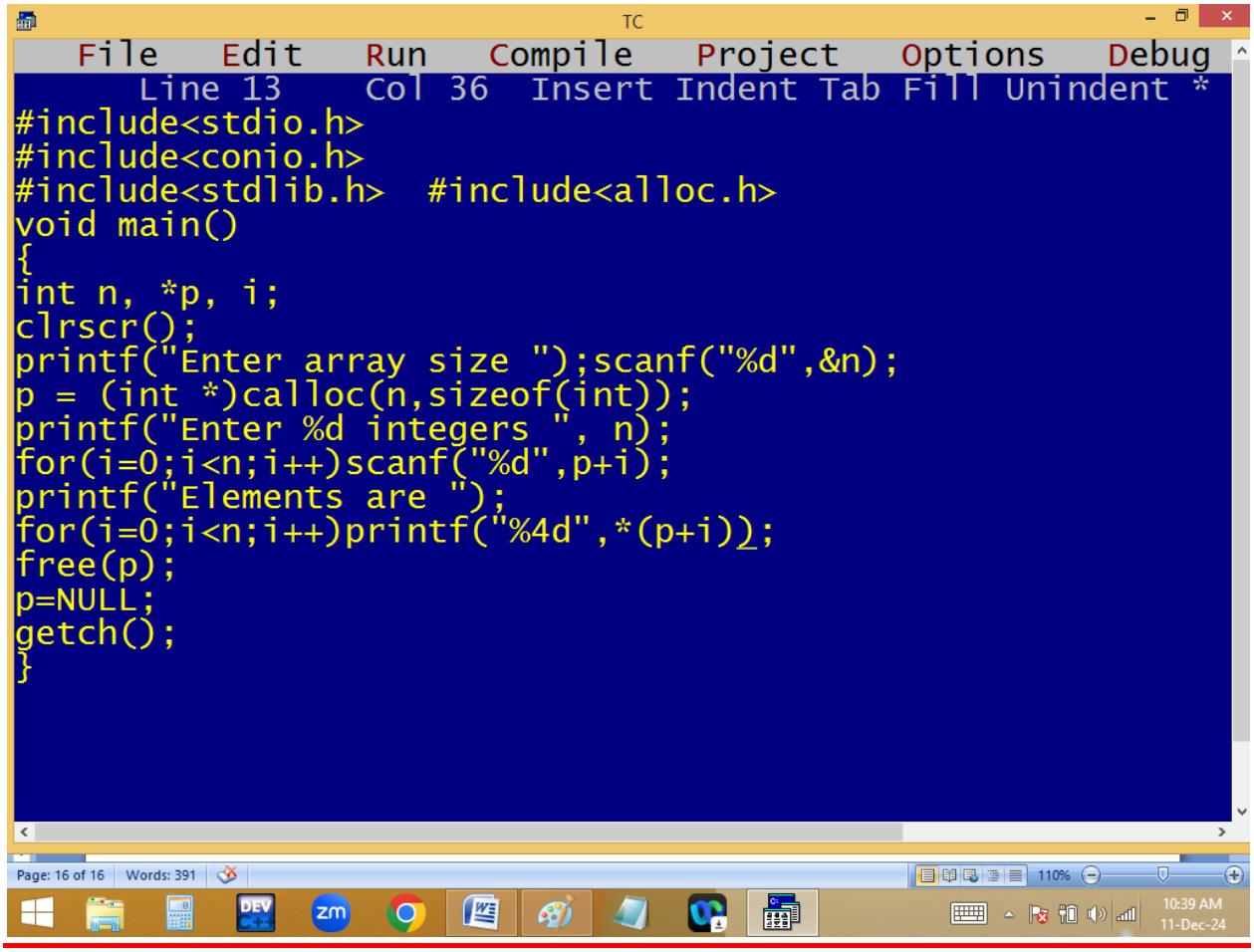
Creating dynamic one-dimensional array:

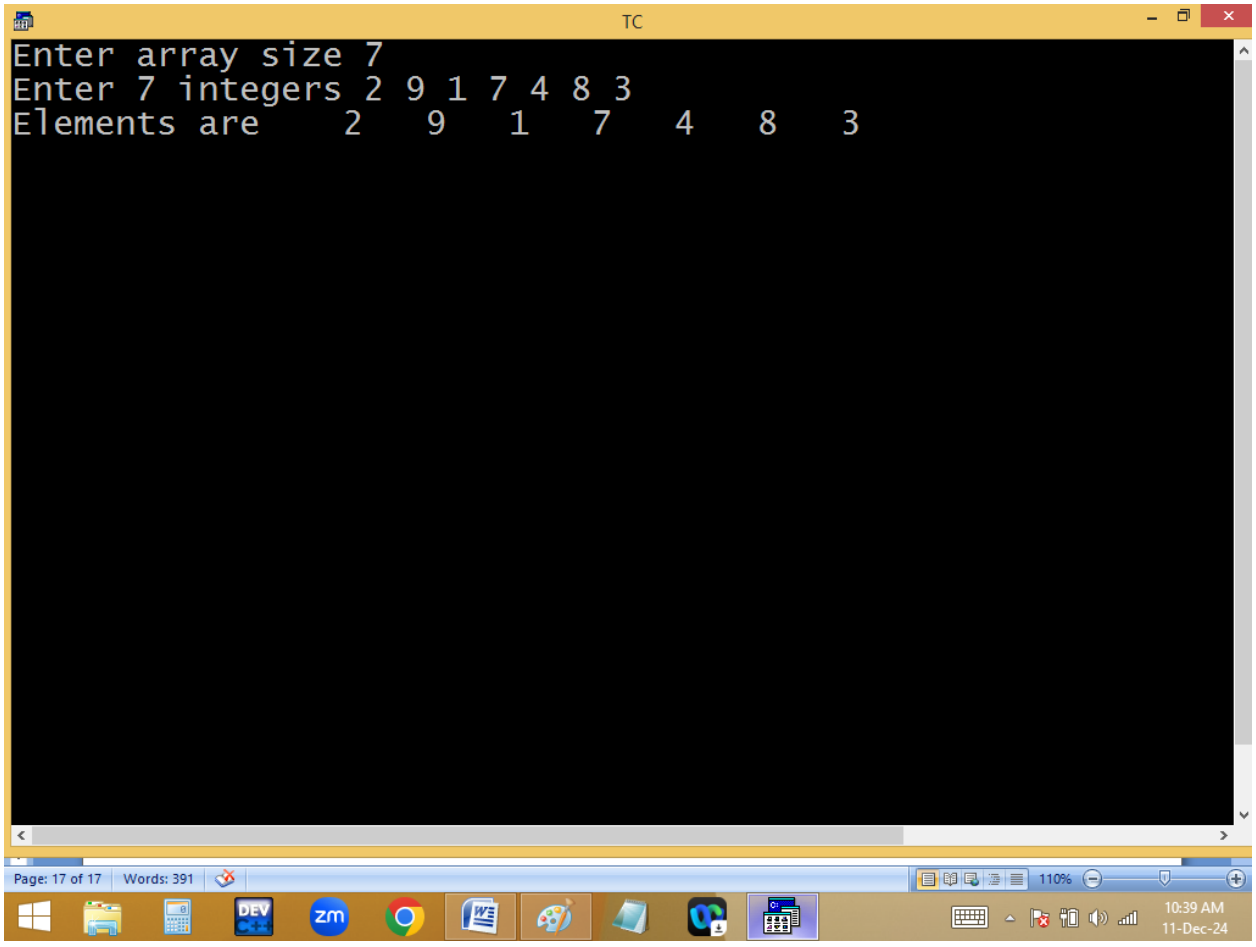



TC
Enter array size 5
Enter 5 integers 2 0 5 -9 6
Elements are 2 0 5 -9 6
Page: 13 of 13
Words: 391
110%
10:37 AM
11-Dec-24

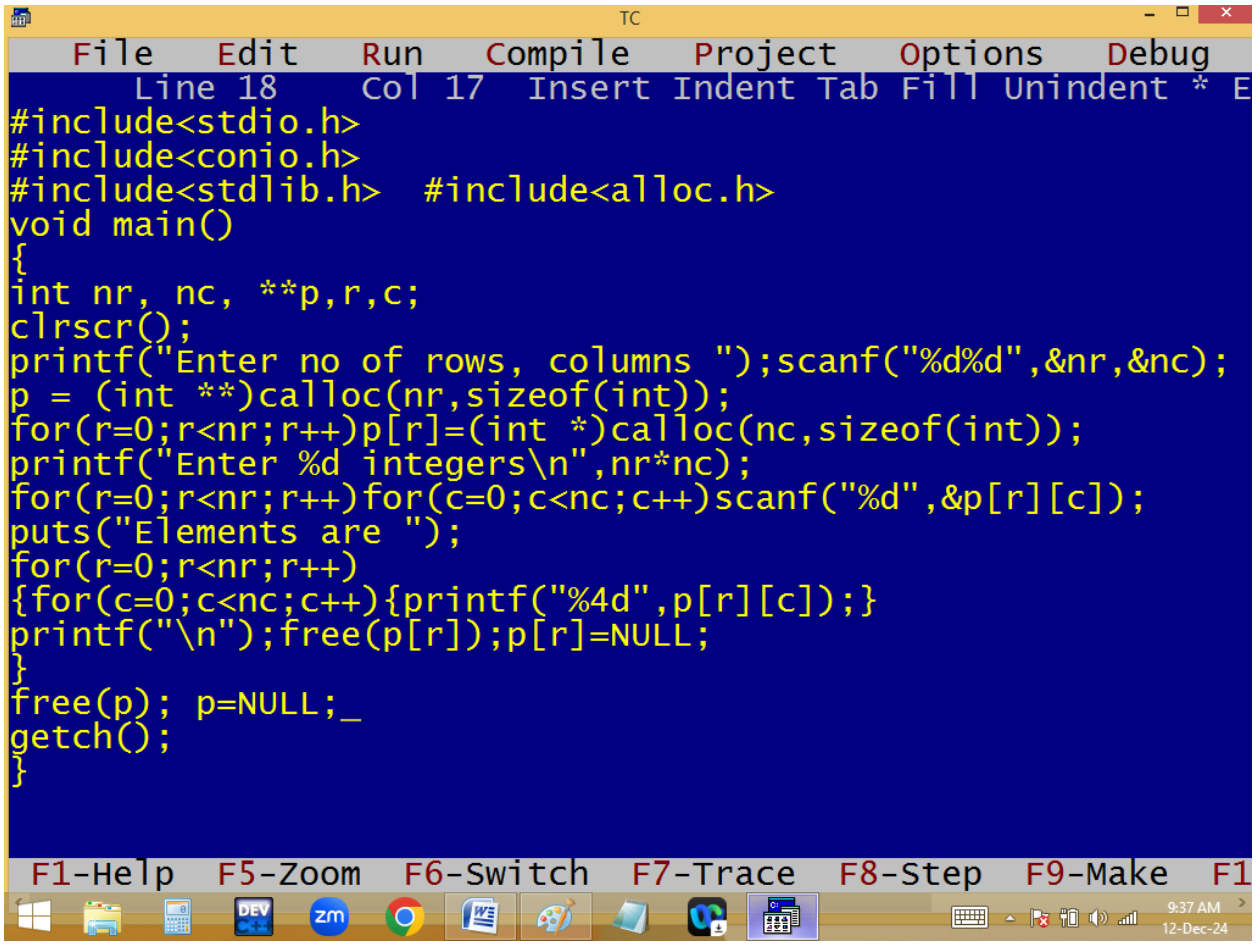

TC
File Edit Run Compile Project Options Debug
Line 9 Col 21 Insert Indent Tab Fill Unindent *
#include<stdio.h>
#include<conio.h>
#include<stdlib.h> #include<alloc.h>
void main()
{
int n, *p, i;
clrscr();
printf("Enter array size ");scanf("%d",&n);
p = (int *)calloc(n,sizeof(int));
printf("Enter %d integers ", n);
for(i=0;i<n;i++)scanf("%d",&p[i]);
printf("Elements are ");
for(i=0;i<n;i++)printf("%4d",p[i]);
free(p);
p=NULL;
getch();
}
Page: 15 of 15
Words: 391
110%
10:38 AM
11-Dec-24

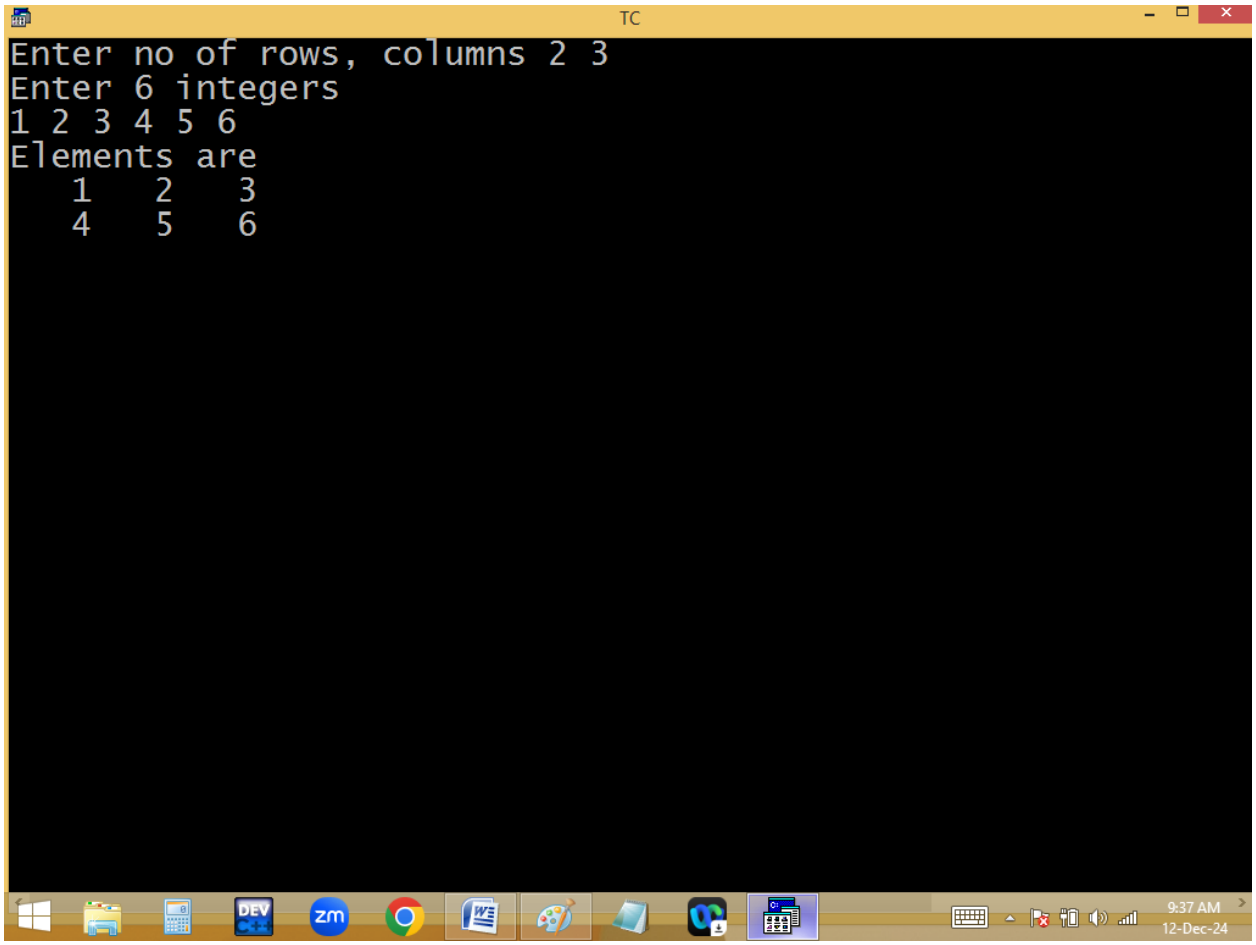

TC
Enter array size 4
Enter 4 integers 2 0 1 3
Elements are 2 0 1 3
Page: 15 of 15
Words: 391
110%
10:38 AM
11-Dec-24

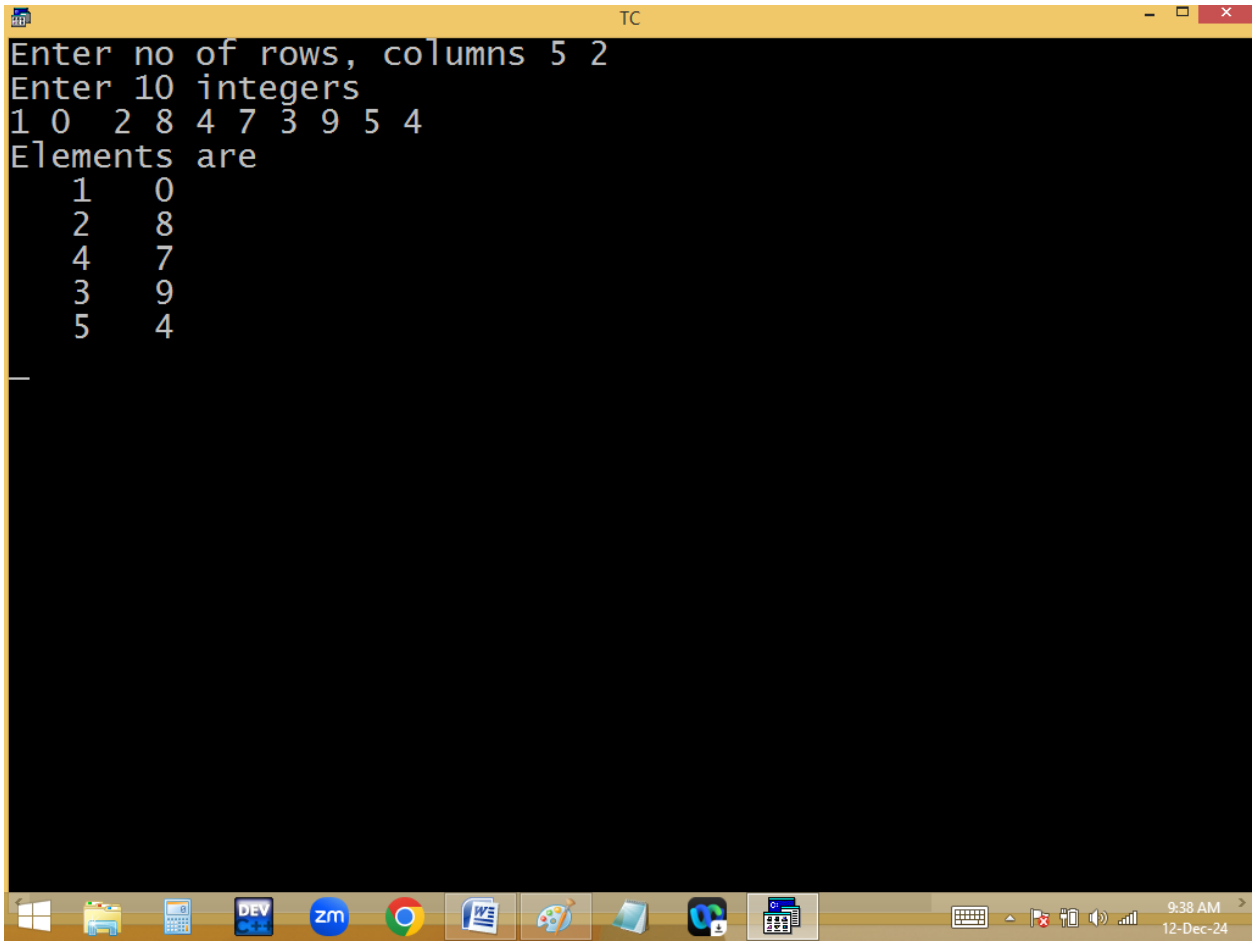

TC
File Edit Run Compile Project Options Debug
Line 13 Col 36 Insert Indent Tab Fill Unindent *
#include<stdio.h>
#include<conio.h>
#include<stdlib.h> #include<alloc.h>
void main()
{
int n, *p, i;
clrscr();
printf("Enter array size ");scanf("%d",&n);
p = (int *)calloc(n,sizeof(int));
printf("Enter %d integers ", n);
for(i=0;i<n;i++)scanf("%d",p+i);
printf("Elements are ");
for(i=0;i<n;i++)printf("%4d",*(p+i));
free(p);
p=NULL;
getch();
}
Page: 16 of 16
Words: 391
110%
10:39 AM
11-Dec-24

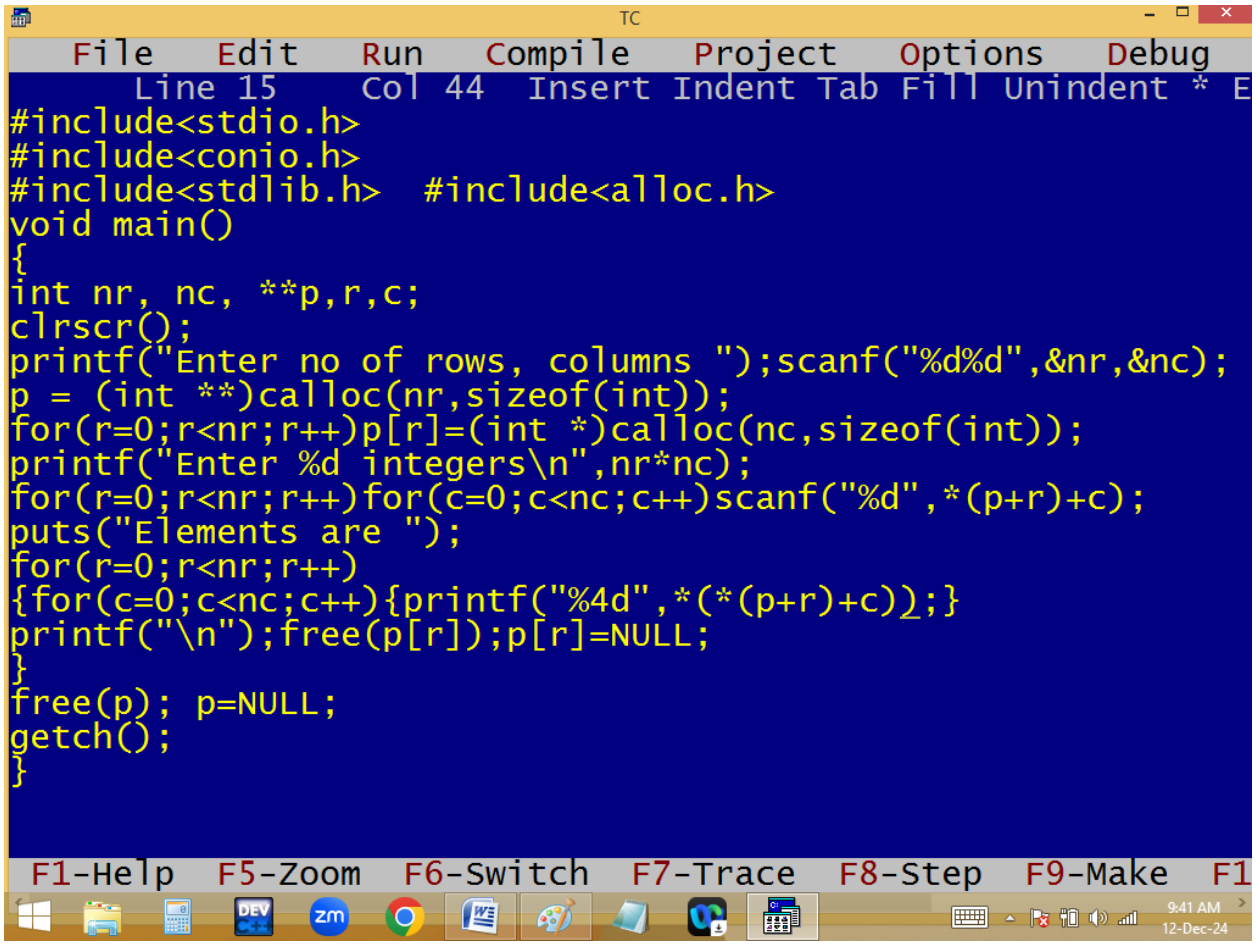

TC
Enter array size 7
Enter 7 integers 2 9 1 7 4 8 3
Elements are 2 9 1 7 4 8 3
Page: 17 of 17
Words: 391
110%
10:39 AM
11-Dec-24

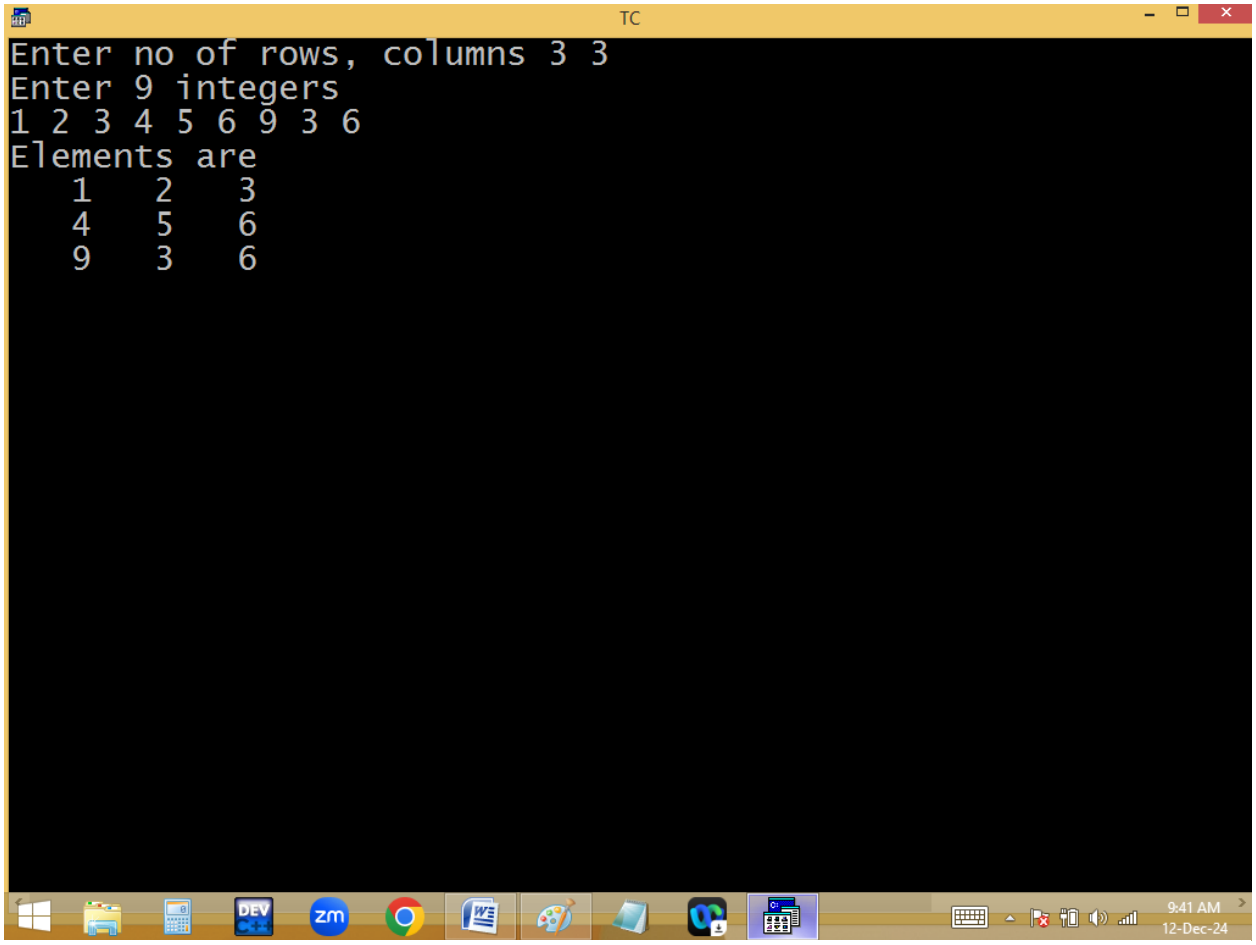
Creating a dynamic multi dimensional array:

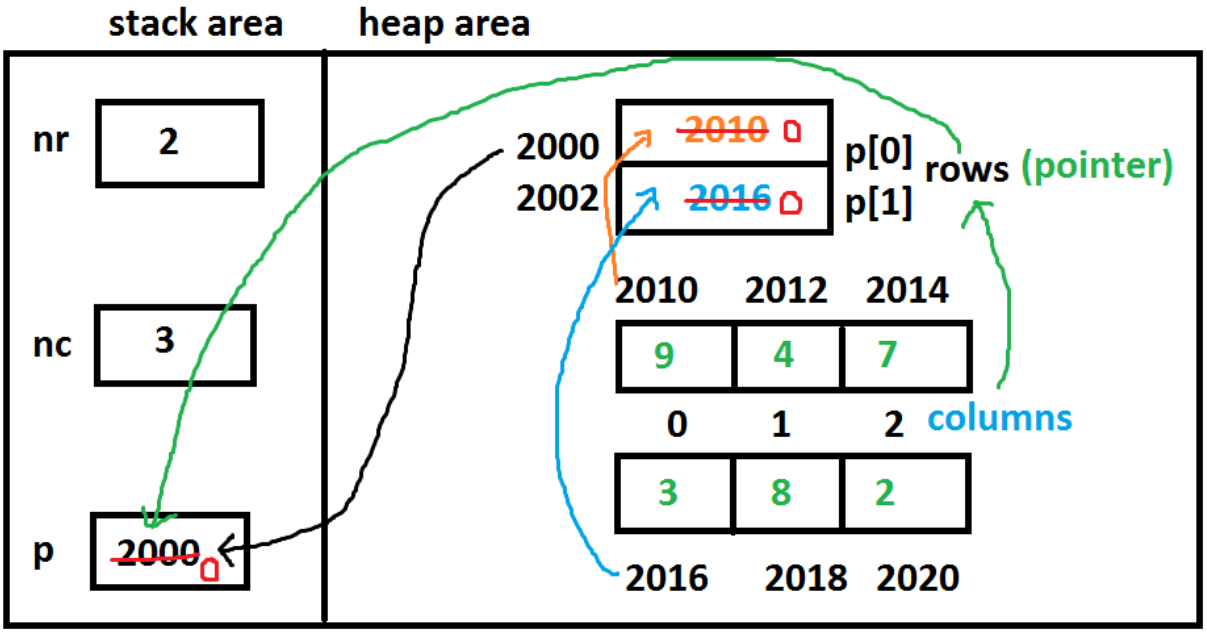

TC
File Edit Run Compile Project Options Debug
Line 18 Col 17 Insert Indent Tab Fill Unindent * E
#include<stdio.h>
#include<conio.h>
#include<stdlib.h> #include<alloc.h>
void main()
{
int nr, nc, **p,r,c;
clrscr();
printf("Enter no of rows, columns ");scanf("%d%d",&nr,&nc);
p = (int **)calloc(nr,sizeof(int));
for(r=0;r<nr;r++)p[r]=(int *)calloc(nc,sizeof(int));
printf("Enter %d integers\n",nr*nc);
for(r=0;r<nr;r++)for(c=0;c<nc;c++)scanf("%d",&p[r][c]);
puts("Elements are ");
for(r=0;r<nr;r++)
{for(c=0;c<nc;c++){printf("%4d",p[r][c]);}
printf("\n");free(p[r]);p[r]=NULL;
}
free(p); p=NULL;
getch();
}
F1-Help F5-Zoom F6-Switch F7-Trace F8-Step F9-Make F1
9:37 AM
12-Dec-24


TC
Enter no of rows, columns 2 3
Enter 6 integers
1 2 3 4 5 6
Elements are
1 2 3
4 5 6
9:37 AM
12-Dec-24

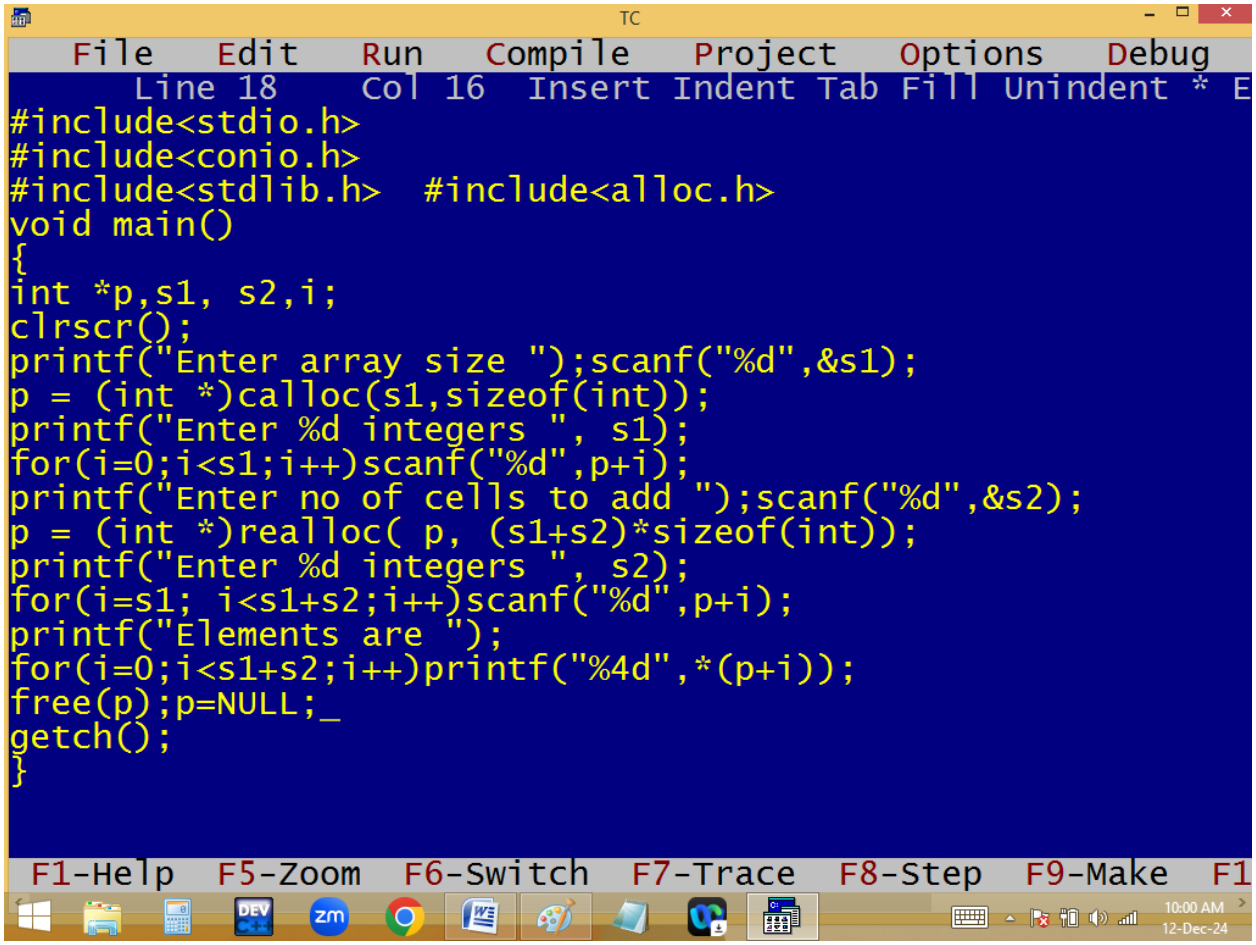

TC
Enter no of rows, columns 5 2
Enter 10 integers
1 0 2 8 4 7 3 9 5 4
Elements are
1 0
2 8
4 7
3 9
5 4
9:38 AM
12-Dec-24

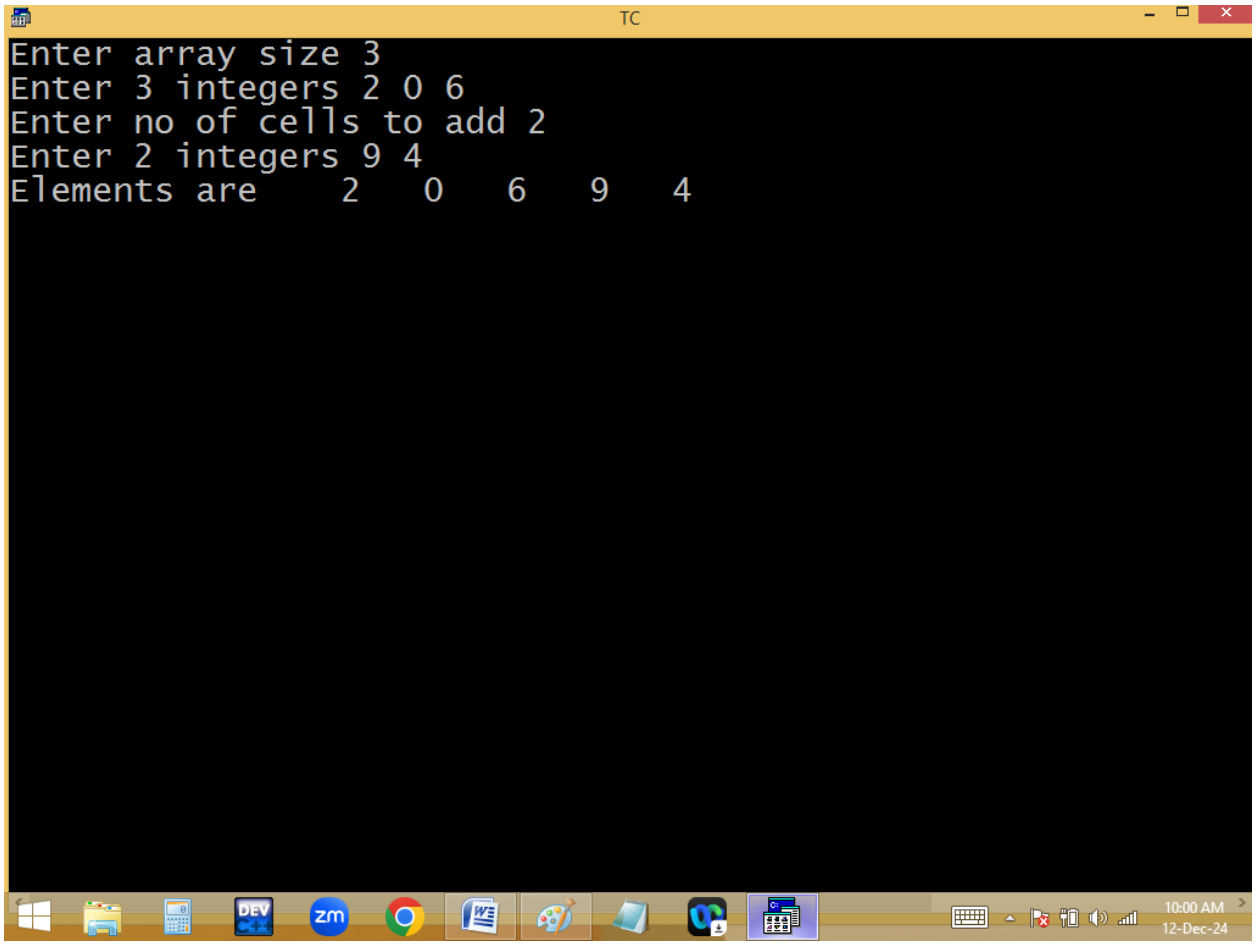

TC
File Edit Run Compile Project Options Debug
Line 15 Col 44 Insert Indent Tab Fill Unindent * E
#include<stdio.h>
#include<conio.h>
#include<stdlib.h> #include<alloc.h>
void main()
{
int nr, nc, **p,r,c;
clrscr();
printf("Enter no of rows, columns ");scanf("%d%d",&nr,&nc);
p = (int **)calloc(nr,sizeof(int));
for(r=0;r<nr;r++)p[r]=(int *)calloc(nc,sizeof(int));
printf("Enter %d integers\n",nr*nc);
for(r=0;r<nr;r++)for(c=0;c<nc;c++)scanf("%d",*(p+r)+c);
puts("Elements are ");
for(r=0;r<nr;r++)
{for(c=0;c<nc;c++){printf("%4d",*(*(p+r)+c));}
printf("\n");free(p[r]);p[r]=NULL;
}
free(p); p=NULL;
getch();
}
F1-Help F5-Zoom F6-Switch F7-Trace F8-Step F9-Make F1
9:41 AM
12-Dec-24

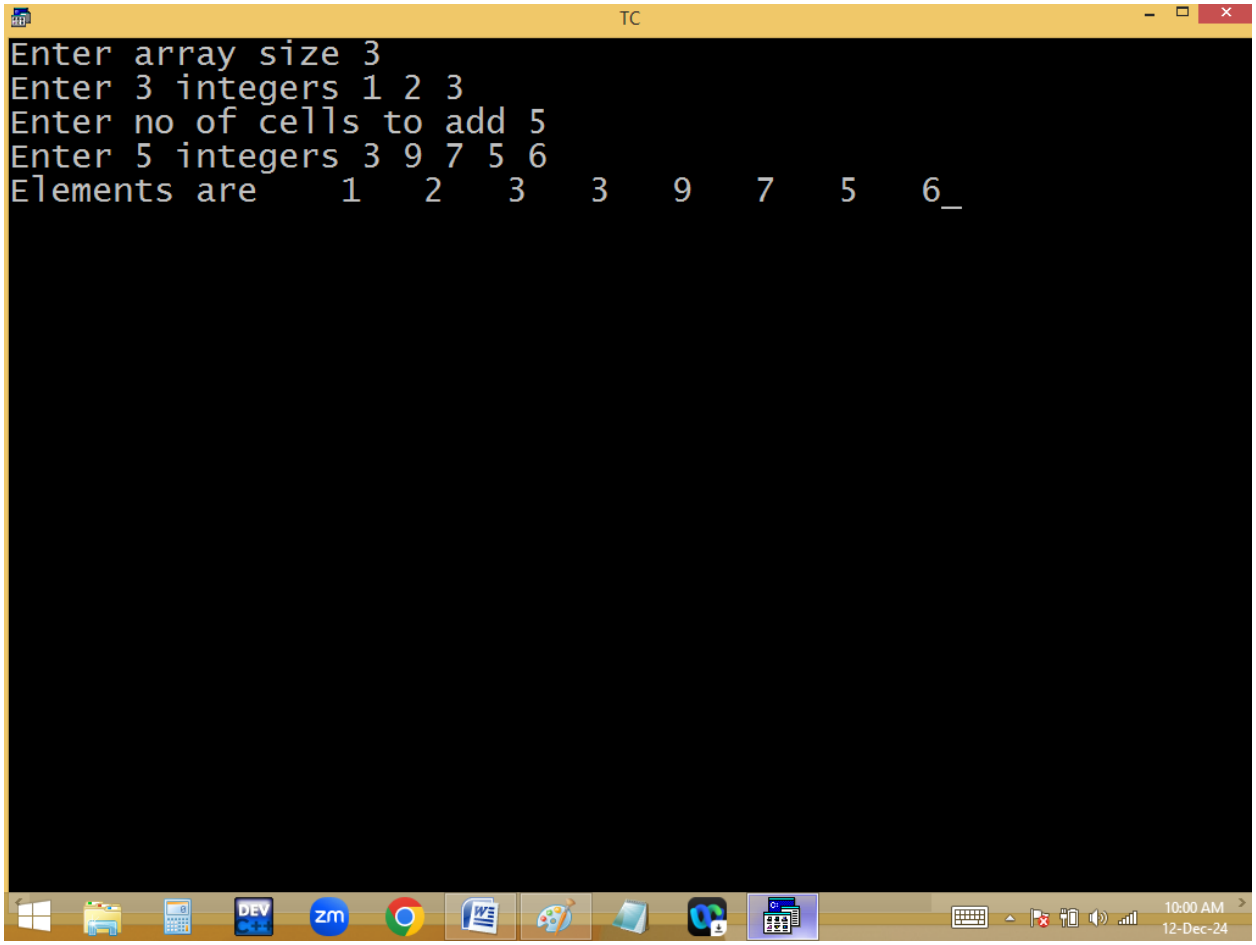

TC
Enter no of rows, columns 3 3
Enter 9 integers
1 2 3 4 5 6 9 3 6
Elements are
1 2 3
4 5 6
9 3 6
9:41 AM
12-Dec-24

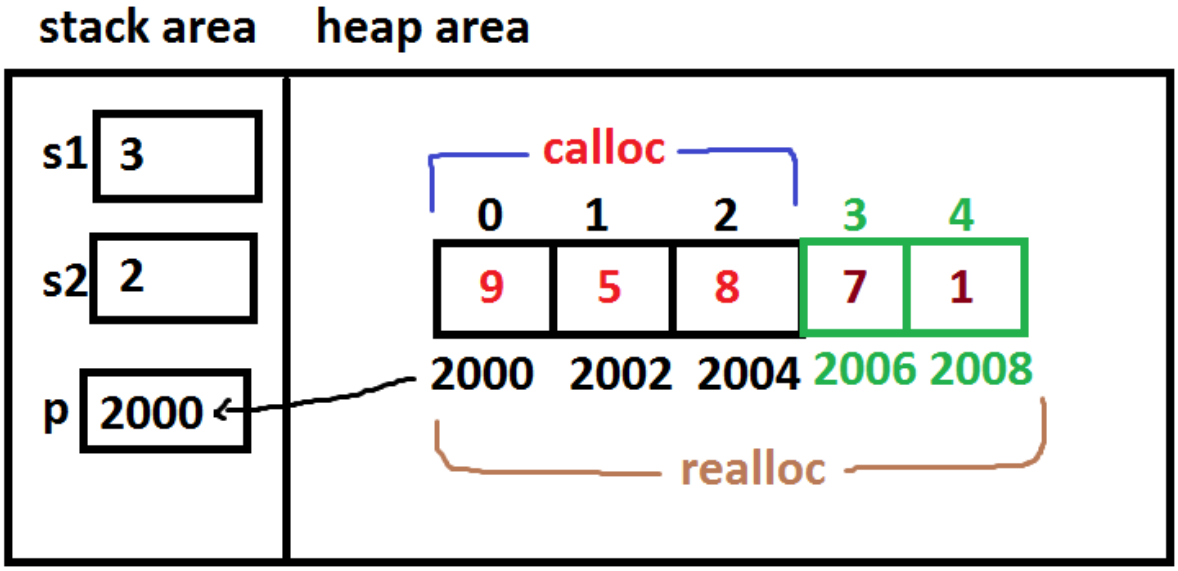

stack area
heap area
nr
2
nc
3
p
2000
2000
2002
2010
2016
p[0]
p[1]
rows (pointer)
2010 2012 2014
9 4 7
0 1 2
columns
3 8 2
2016 2018 2020

realloc():


TC
File Edit Run Compile Project Options Debug
Line 18 Col 16 Insert Indent Tab Fill Unindent * E
#include<stdio.h>
#include<conio.h>
#include<stdlib.h> #include<alloc.h>
void main()
{
int *p,s1, s2,i;
clrscr();
printf("Enter array size ");scanf("%d",&s1);
p = (int *)calloc(s1,sizeof(int));
printf("Enter %d integers ", s1);
for(i=0;i<s1;i++)scanf("%d",p+i);
printf("Enter no of cells to add ");scanf("%d",&s2);
p = (int *)realloc(p, (s1+s2)*sizeof(int));
printf("Enter %d integers ", s2);
for(i=s1; i<s1+s2;i++)scanf("%d",p+i);
printf("Elements are ");
for(i=0;i<s1+s2;i++)printf("%4d",*(p+i));
free(p);p=NULL;_
getch();
}
F1-Help F5-Zoom F6-Switch F7-Trace F8-Step F9-Make F1
10:00 AM
12-Dec-24


TC
Enter array size 3
Enter 3 integers 2 0 6
Enter no of cells to add 2
Enter 2 integers 9 4
Elements are 2 0 6 9 4
10:00 AM
12-Dec-24


TC
Enter array size 3
Enter 3 integers 1 2 3
Enter no of cells to add 5
Enter 5 integers 3 9 7 5 6
Elements are 1 2 3 3 9 7 5 6
10:00 AM
12-Dec-24


stack area
heap area
s1
3
s2
2
p
2000
calloc
0
1
2
3
4
9
5
8
7
1
2000
2002
2004
2006
2008
realloc

USER DEFINED FUNCTIONS

What is a function?

It is a small program used to do a particular task.

It is a sub prog / sub routine / procedure / module / structure

It is a reusable code component.

It is a self contained block.

It is a small program within another program.

Adv:

1. **Modularity: dividing big program into small modules as per the project requirement.**
2. **Simplicity: easy to read and understand**
3. **Reusability: write once, use many times.**
4. **Efficiency: performance is high**

Marriage function

Every function is divided into 3 parts

1. Function declaration / proto type: before main() or within main()
   Eg: void sum();
2. Function calling – within main() only
   Eg: sum();
3. Function definition – outside main() only

USER DEFINED FUNCTIONS

What is a function?

1. It is a small program used to do a particular task.
2. It is a small program with in another program.
3. It is a sub program or sub routine.
4. It is a procedure.
5. It is a self-contained block.
6. It is a reusable code component.
7. It is a module.

Advantages:

1. **Modularity**: Dividing program instructions into small pieces.
2. **Simplicity**: It is easy to understand the program instructions.
3. **Reusability**: Just write once, use several times.
4. **Efficiency**: performance is improved.
5. It is easy to identify the errors.

Entire ‘C’ program is collection of functions. Hence ‘C’ is a function oriented structured programming language.

We are having 2 types of functions.

1. Predefined / library / built-in functions

These are the functions provided with software and ready to use.

Eg: printf(), scanf(),…

2. **User defined functions**: They are created by the user.

Eg: sum(), prime(), factorial(),……

Every user defined function is divided into 3 parts.

1. Function declaration/ proto typing
2. Function calling
3. Function definition

Function declaration/proto typing:

Generally function declaration is conducted before or within the main(), before function calling.

It tells the compiler that we are going to use this kind of function in our program in future.

Syntax:

[return_datatype] function_name ([arguments / parameters]);

Eg:

void sum();

Function calling:

It should be conducted within the main() only.

When a function is invoked (called), the compiler will search for the matching function definition and it is available then the program execution is jumped from function calling area to the function definition area.

Linking a function call to the function definition is called **binding.** C language supports only static or compile time binding. But C++ supports static and dynamic binding[run time binding].

Without function call, a function never participates in function execution. Hence it is mandatory to call a function in a program.

Syntax:

function_name([arguments]);

Eg: sum();

Function definition:

It contains the function header and body. The function header should be matched with function declaration.

It is conducted outside the main(). If the definition is conducted before the main(), there is no need of function declaration.

Function header consists of function name and the parameters.

Function body consists of statements related to the task of function.

When a function is invoked, the program execution is shifted from function calling area (main()) to function definition area. After executing the function body, the execution is jumped to the next statement after the function call in main().

Function declaration and function definition should be identical.

Syntax:

Function Header

```
[return data type]  Function_name(
[parameters/arguments] )
{
  Statements;
}
```

Function Body

[return value;]

}

Here return data type indicates the type of value that the function is returning to main() / called function. It is depended on the return value.

If the sub program is having return statement then it is called function and without return value it is called procedure.

The **default return value of a function is integer**. Use the keyword **void** [nothing] when the function is not having any return value. Otherwise integer will become the return value.

A function may have any number of return statements. But only one return statement is executed at a time. The statements after the return statement are not executed and gives compile time warning.

Return data type should be matched with return value data type.

Function name is used to identify a function and it should not be predefined.

Arguments / parameters used to carry the values from function calling area to function definition area. They are optional and of two types.

1. **Actual** arguments/parameters.
2. **Formal** arguments/parameters.

The arguments we are sending in function calling at main() are called actual parameters and the parameters we are using in function definition to receive the actual argument values is called formal parameters.

Actual and formal parameter names may be same or different.

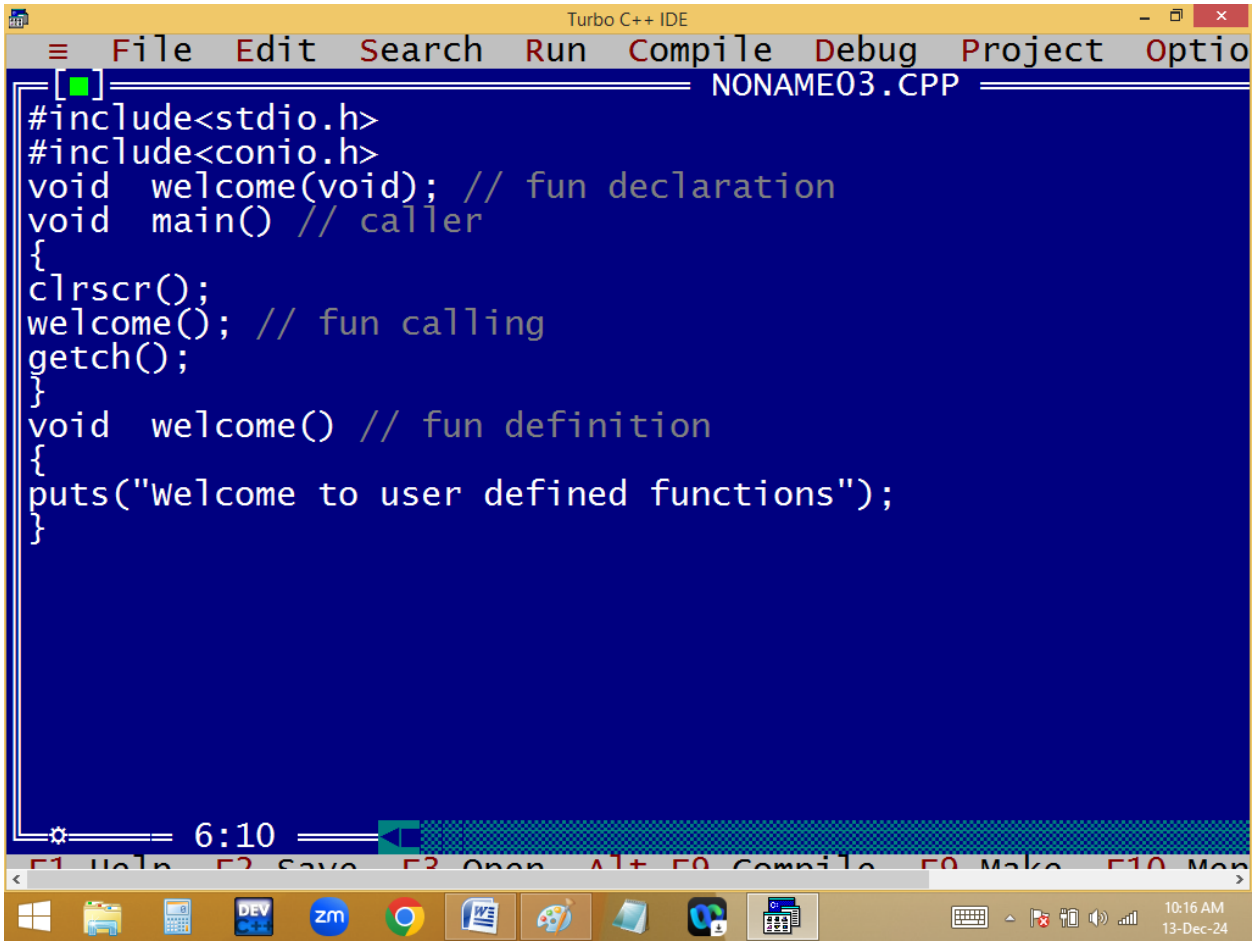
Difference between arguments and parameters is "arguments are nothing but different vehicles which are parked in same parking area and the parking area is the parameters. i.e. arguments are always changed but parameters are fixed.

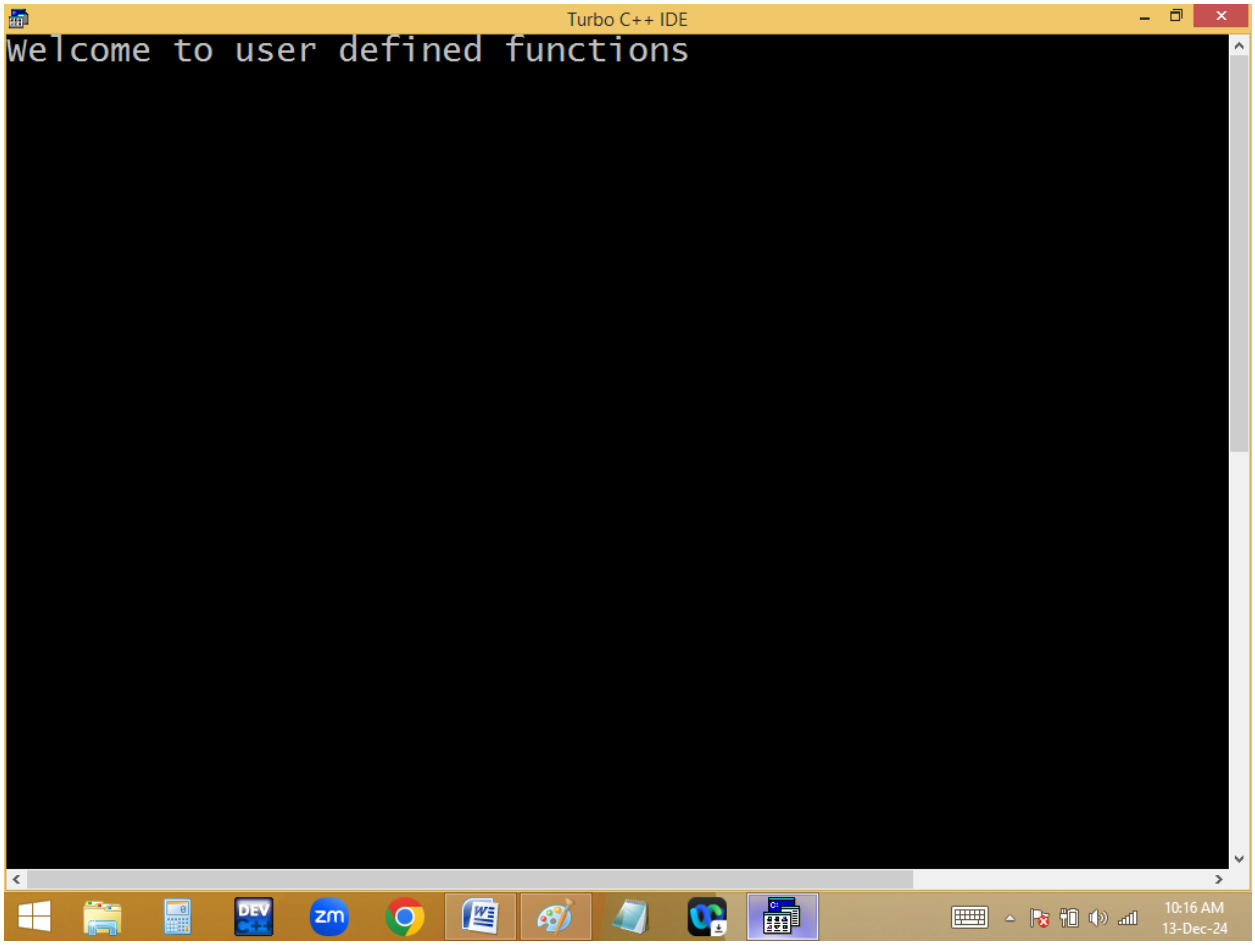
The calling function arguments should be matched with definition parameters list in quantity, data type and order.

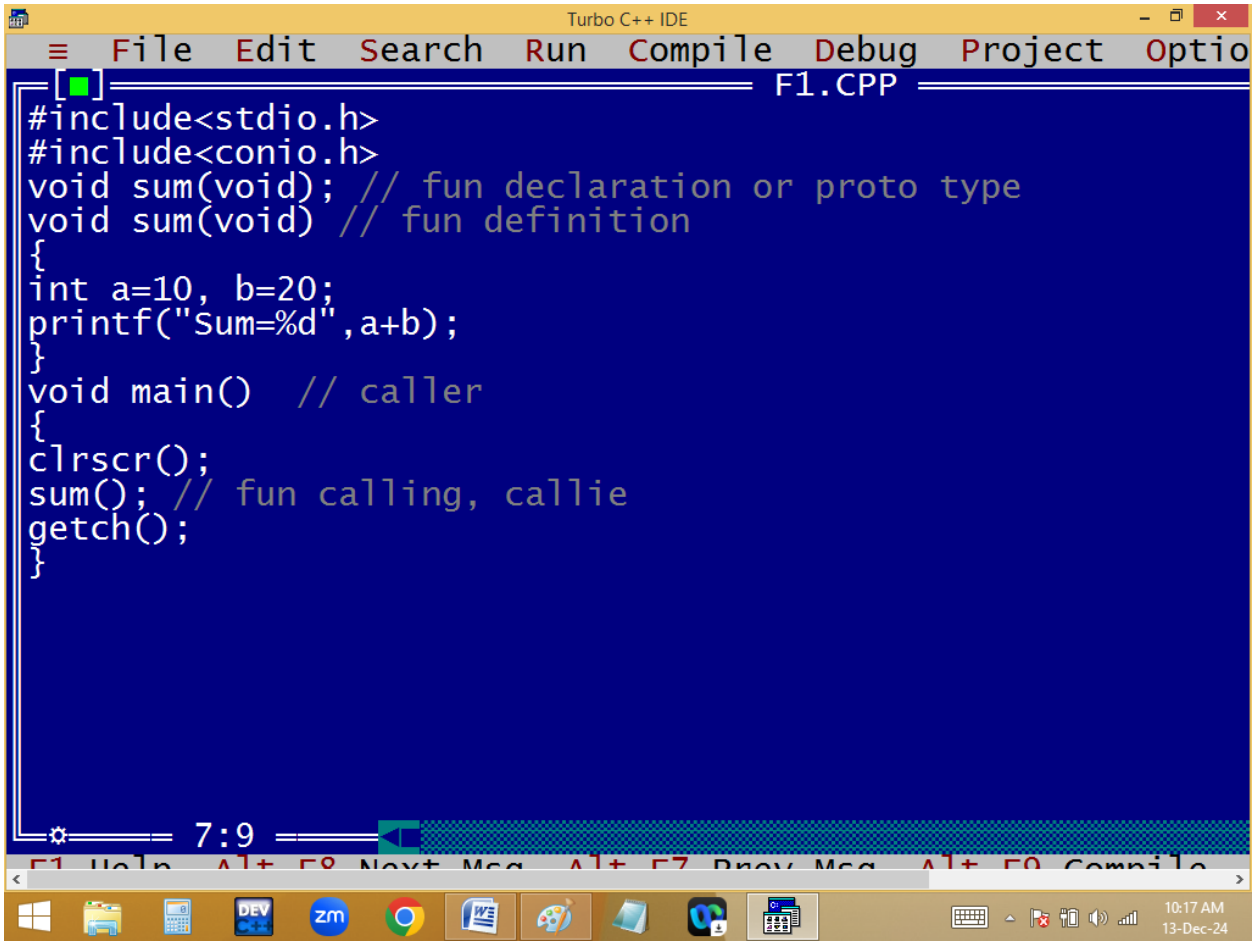
Based on arguments and return values, functions are divided into 4 types.

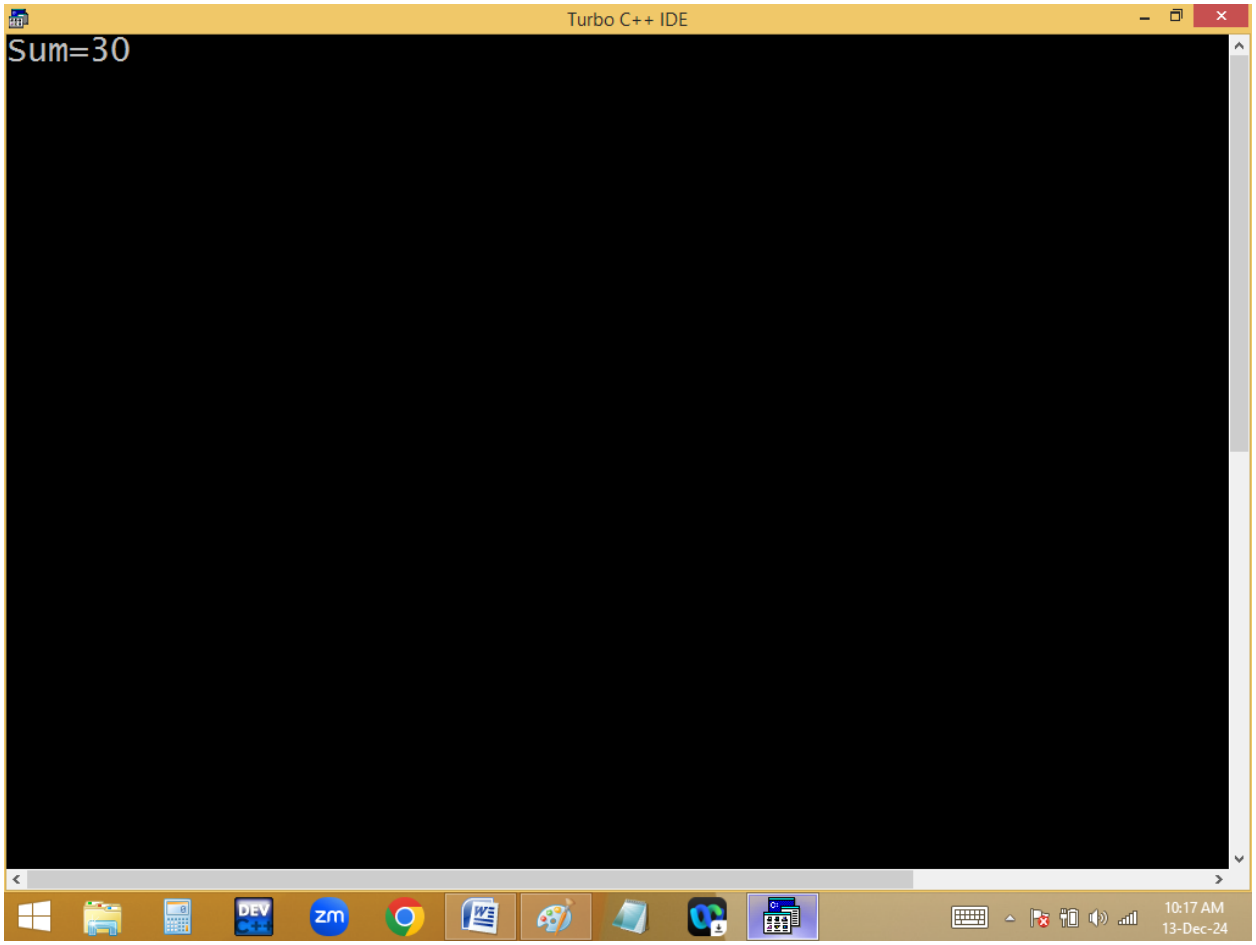
1. Function without arguments, without return values.
2. Function with arguments and with return values.
3. Function without arguments and with return values.
4. Function with arguments and without return values.

Eg: **function without arguments, without return values**

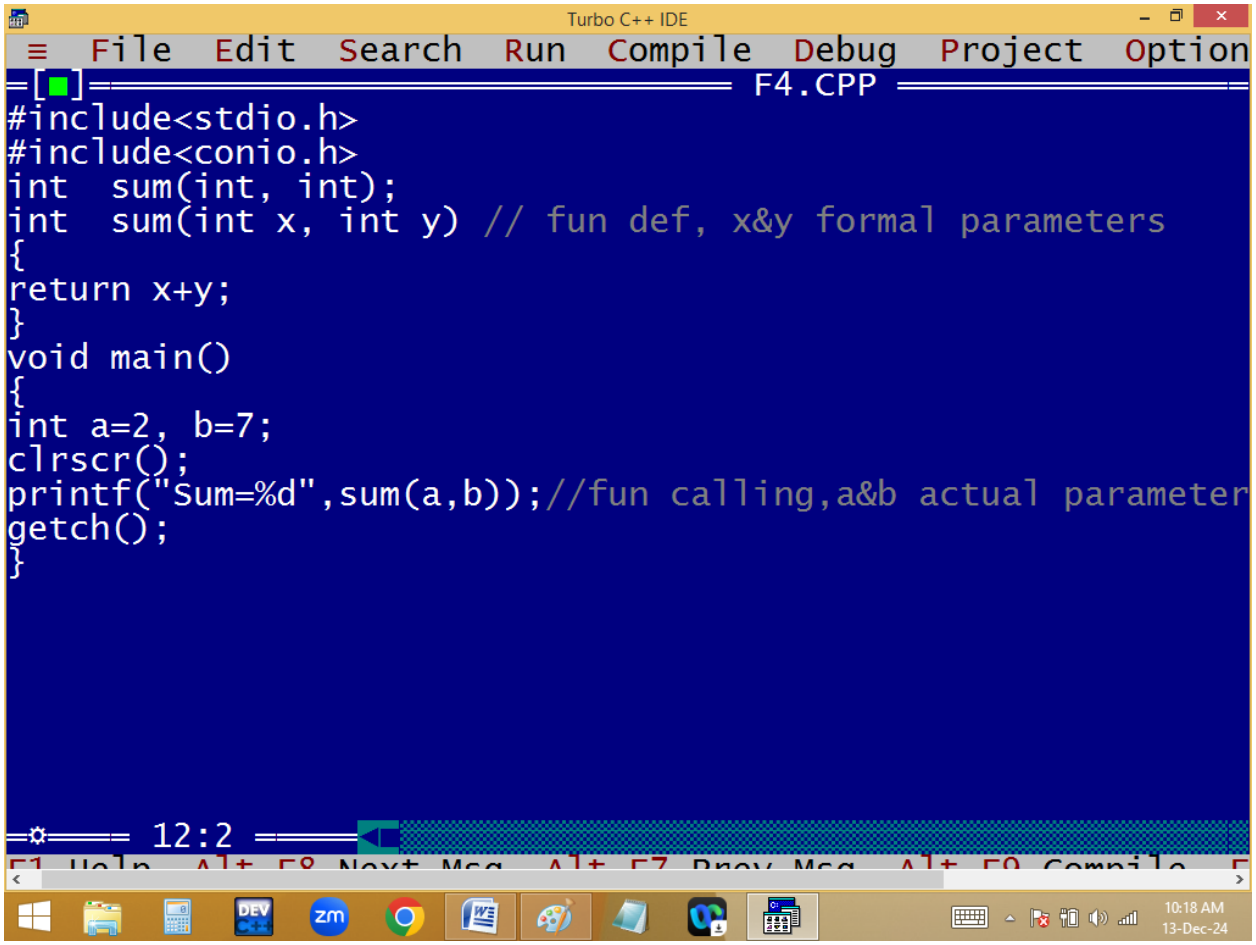

Turbo C++ IDE
File Edit Search Run Compile Debug Project Optio
NONAME03.CPP
#include<stdio.h>
#include<conio.h>
void welcome(void); // fun declaration
void main() // caller
{
clrscr();
welcome(); // fun calling
getch();
}
void welcome() // fun definition
{
puts("Welcome to user defined functions");
}
6:10
10:16 AM
13-Dec-24

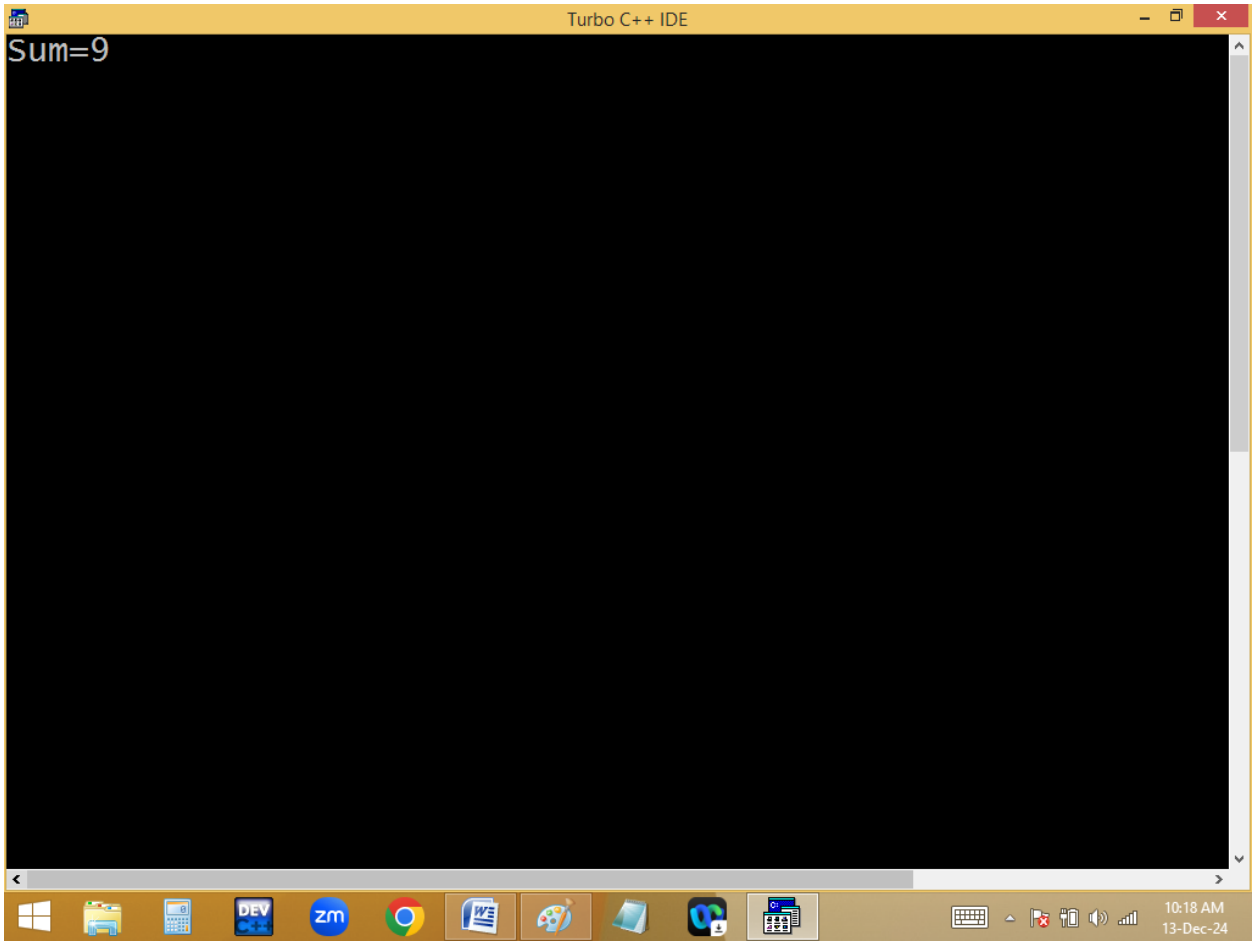

Turbo C++ IDE
Welcome to user defined functions
10:16 AM
13-Dec-24


Turbo C++ IDE
File Edit Search Run Compile Debug Project Optio
F1.CPP
#include<stdio.h>
#include<conio.h>
void sum(void); // fun declaration or proto type
void sum(void) // fun definition
{
int a=10, b=20;
printf("Sum=%d",a+b);
}
void main() // caller
{
clrscr();
sum(); // fun calling, callie
getch();
}
7:9
10:17 AM
13-Dec-24

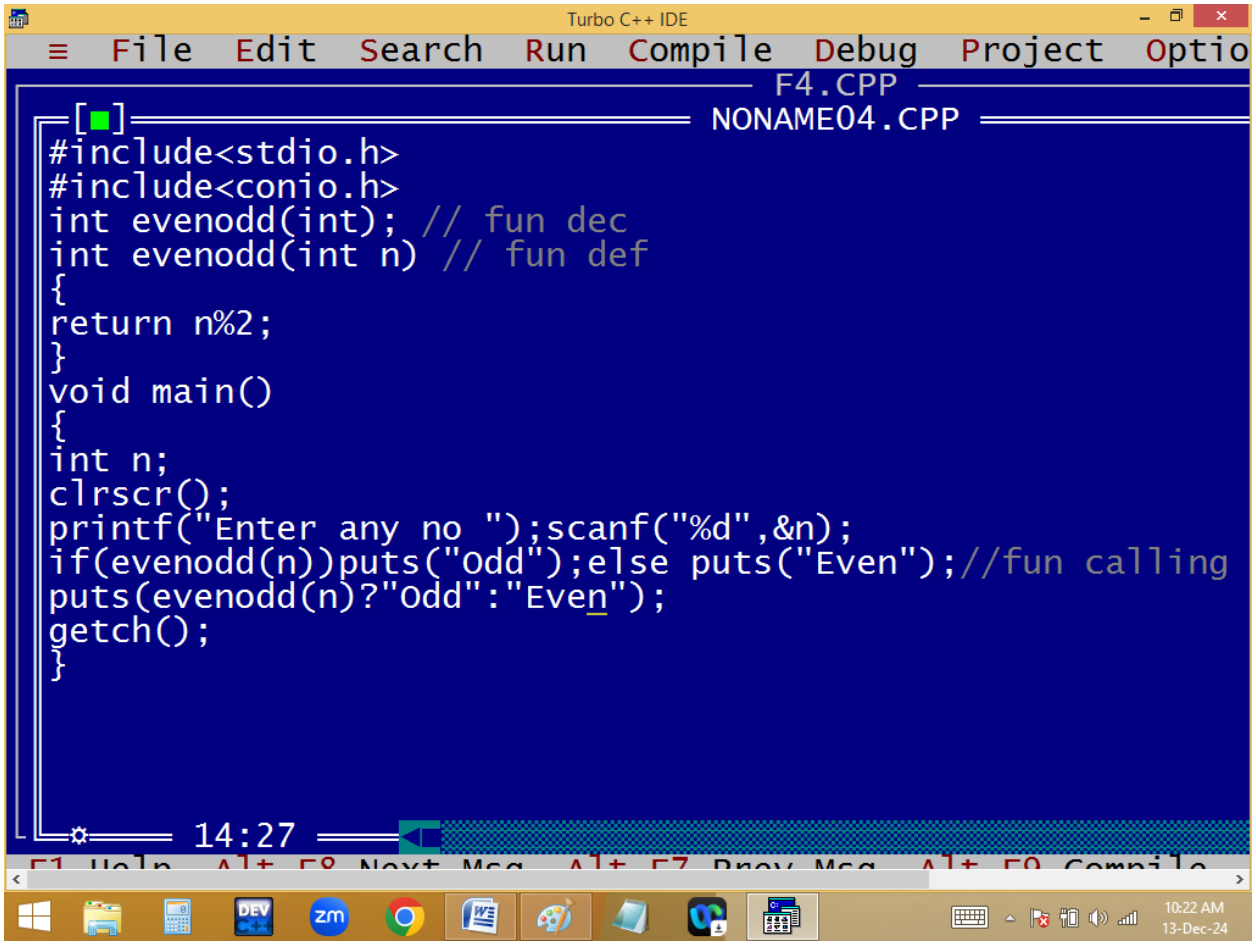


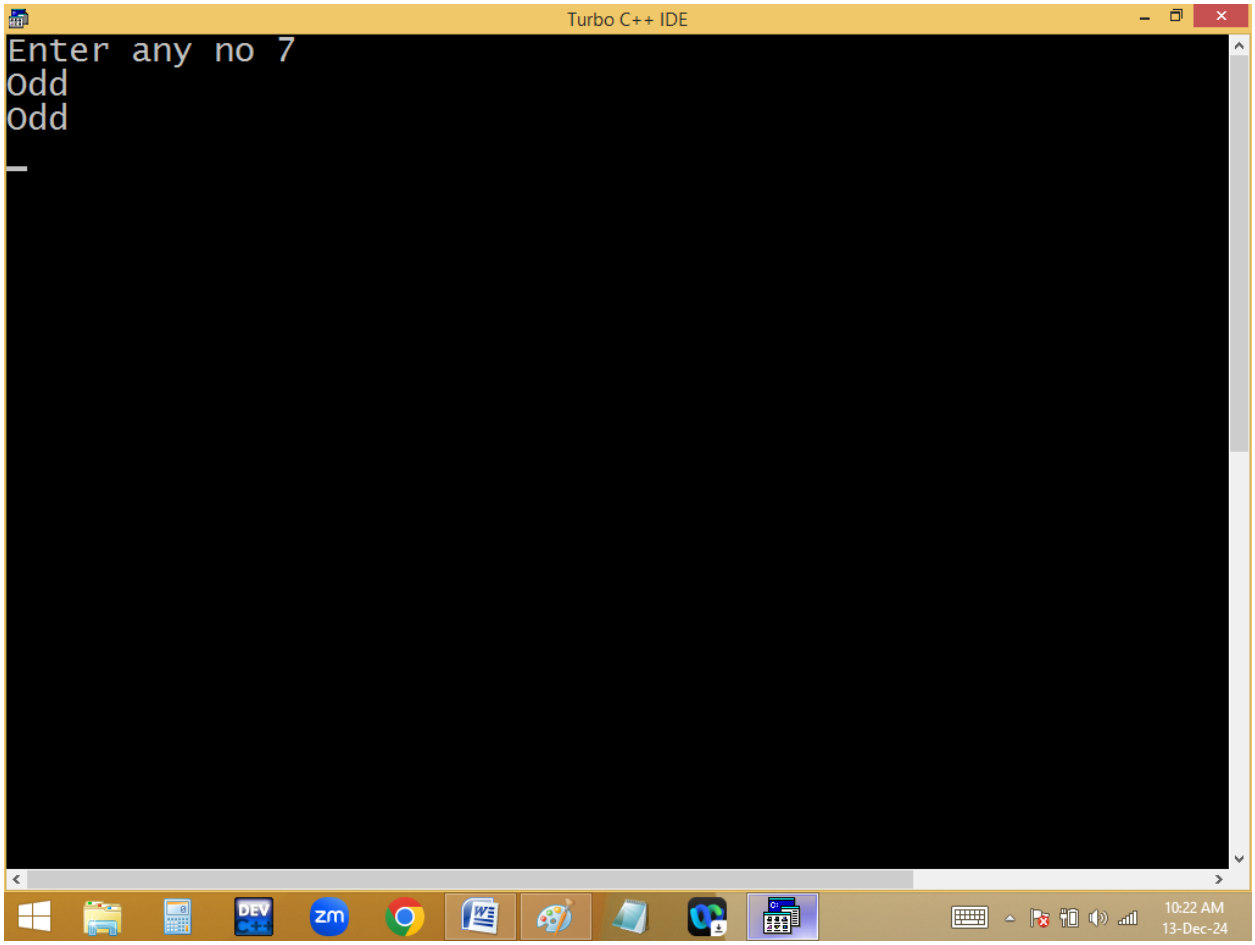
Function with arguments, with return value:

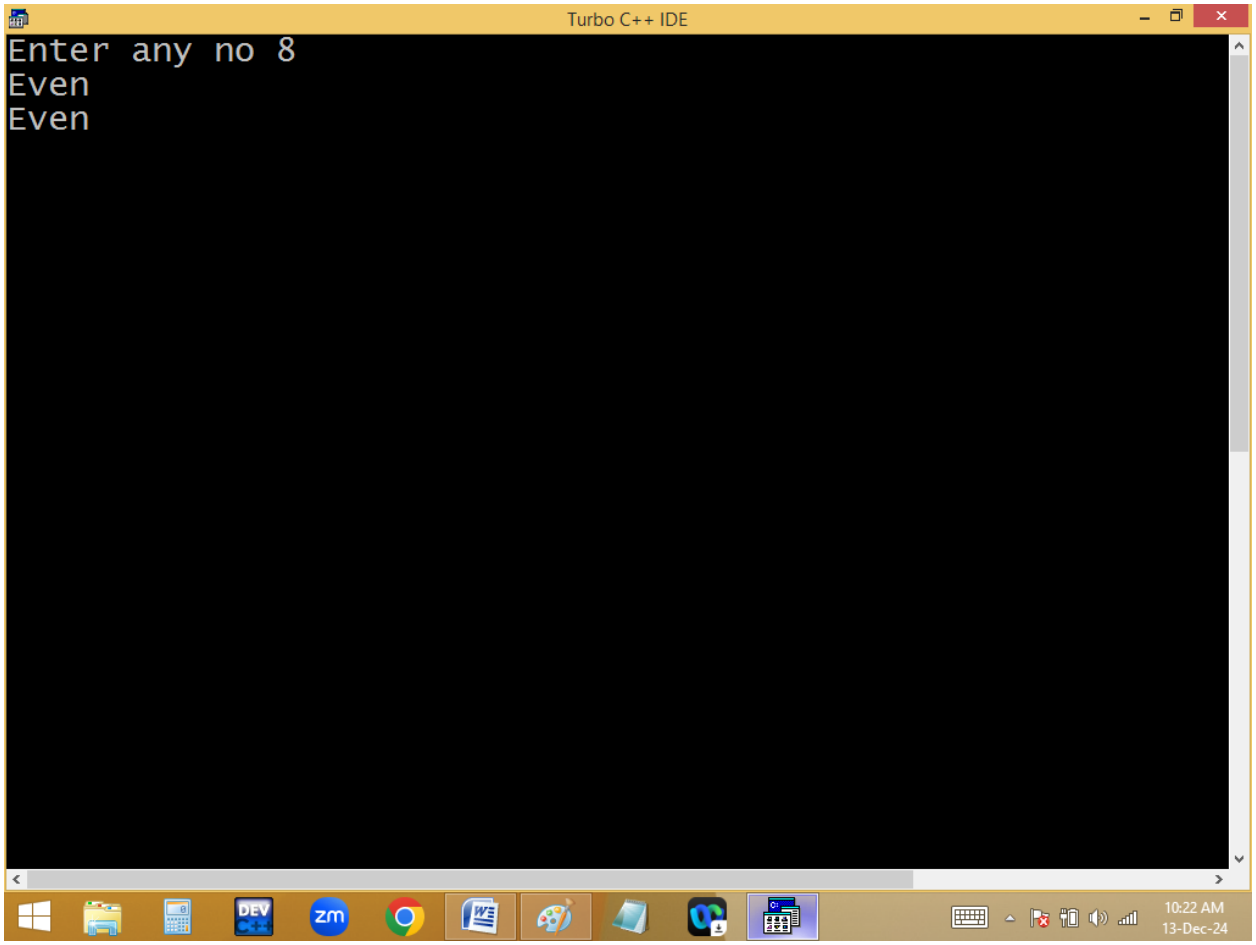

Turbo C++ IDE
File Edit Search Run Compile Debug Project Option
F4.CPP
#include<stdio.h>
#include<conio.h>
int sum(int, int);
int sum(int x, int y) // fun def, x&y formal parameters
{
return x+y;
}
void main()
{
int a=2, b=7;
clrscr();
printf("Sum=%d",sum(a,b));//fun calling,a&b actual parameter
getch();
}
12:2
10:18 AM
13-Dec-24



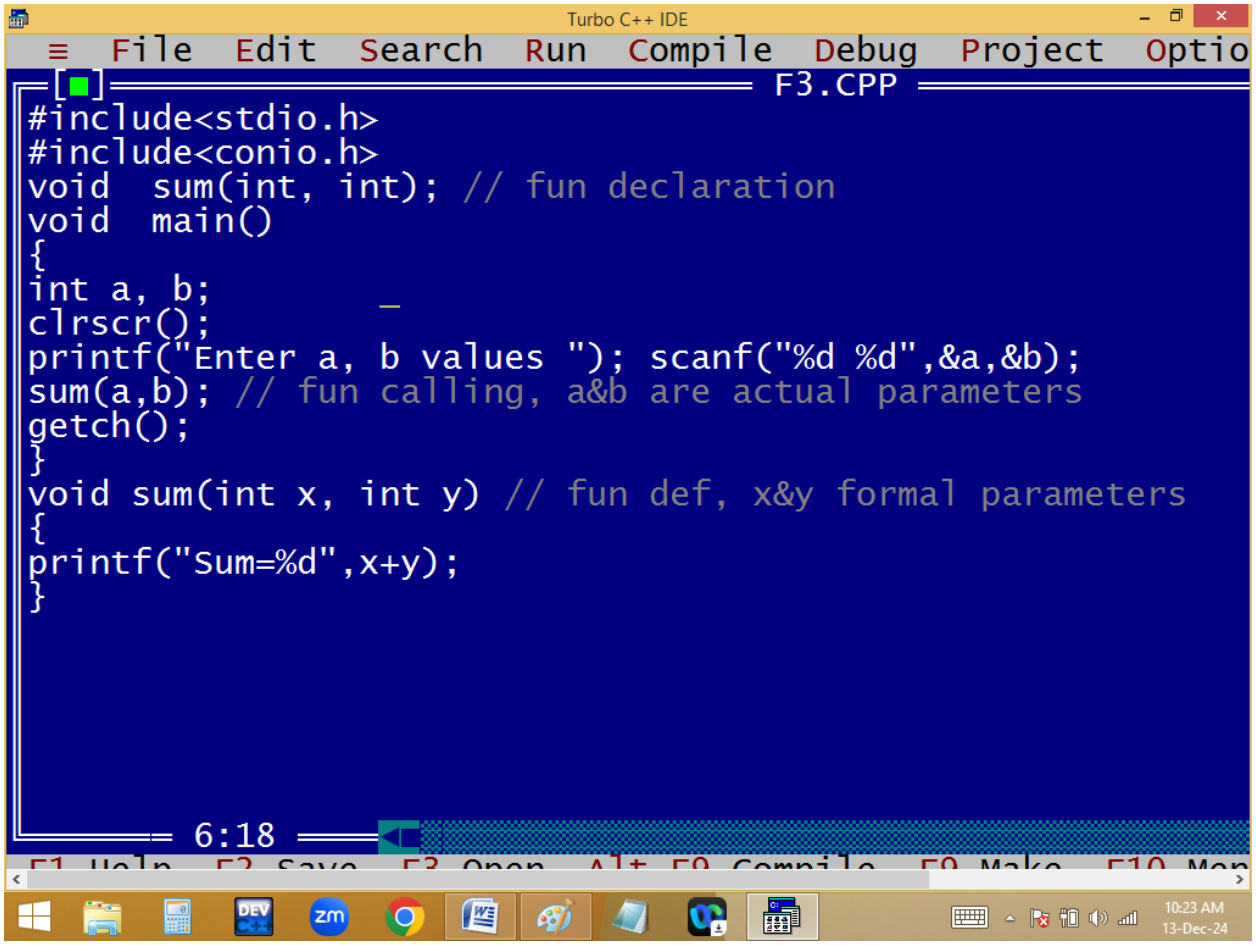
Finding even/odd using user defined function?

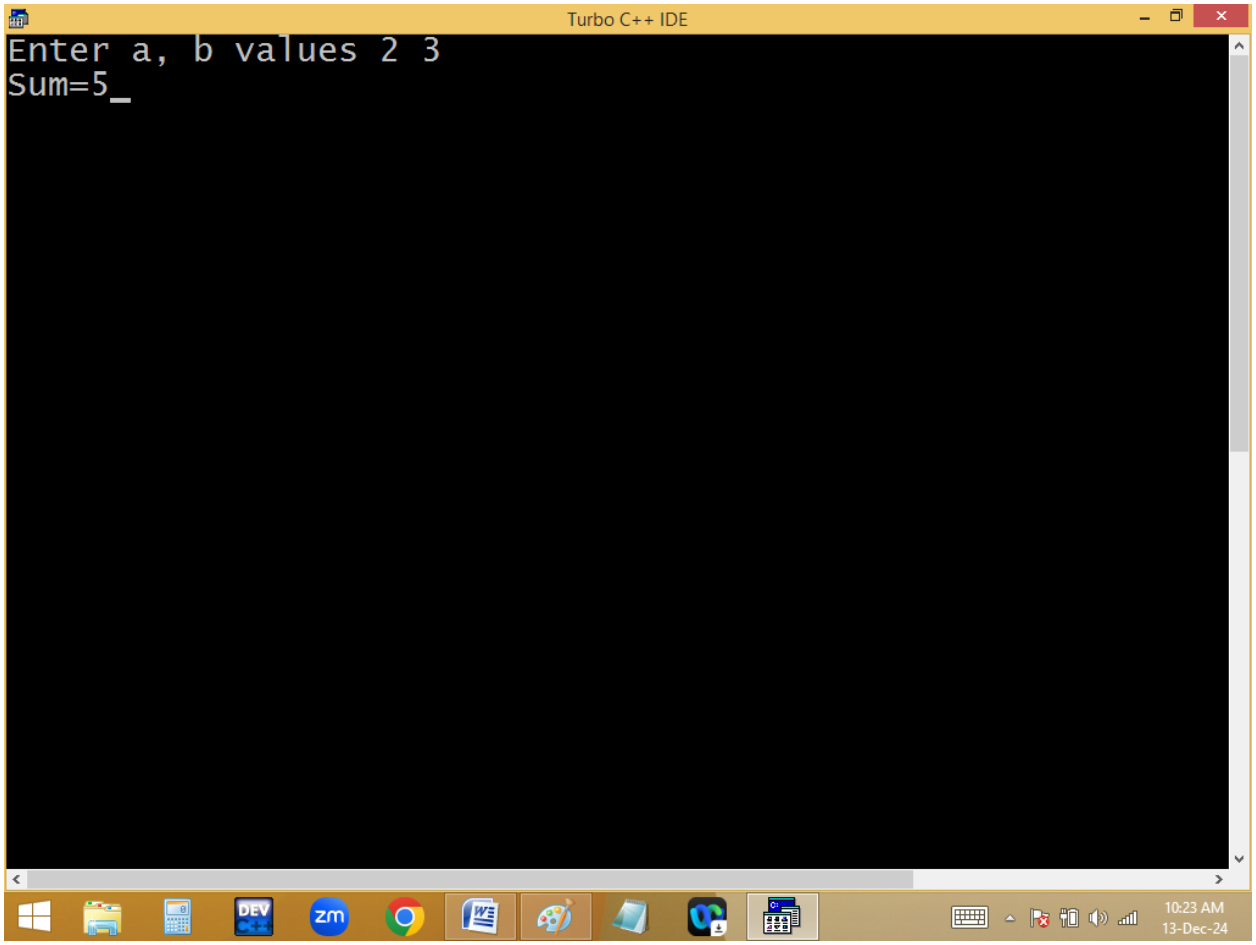

Turbo C++ IDE
File Edit Search Run Compile Debug Project Optio
F4.CPP
NONAME04.CPP
#include<stdio.h>
#include<conio.h>
int evenodd(int); // fun dec
int evenodd(int n) // fun def
{
return n%2;
}
void main()
{
int n;
clrscr();
printf("Enter any no ");scanf("%d",&n);
if(evenodd(n))puts("Odd");else puts("Even");//fun calling
puts(evenodd(n)?"Odd":"Even");
getch();
}
14:27
10:22 AM
13-Dec-24

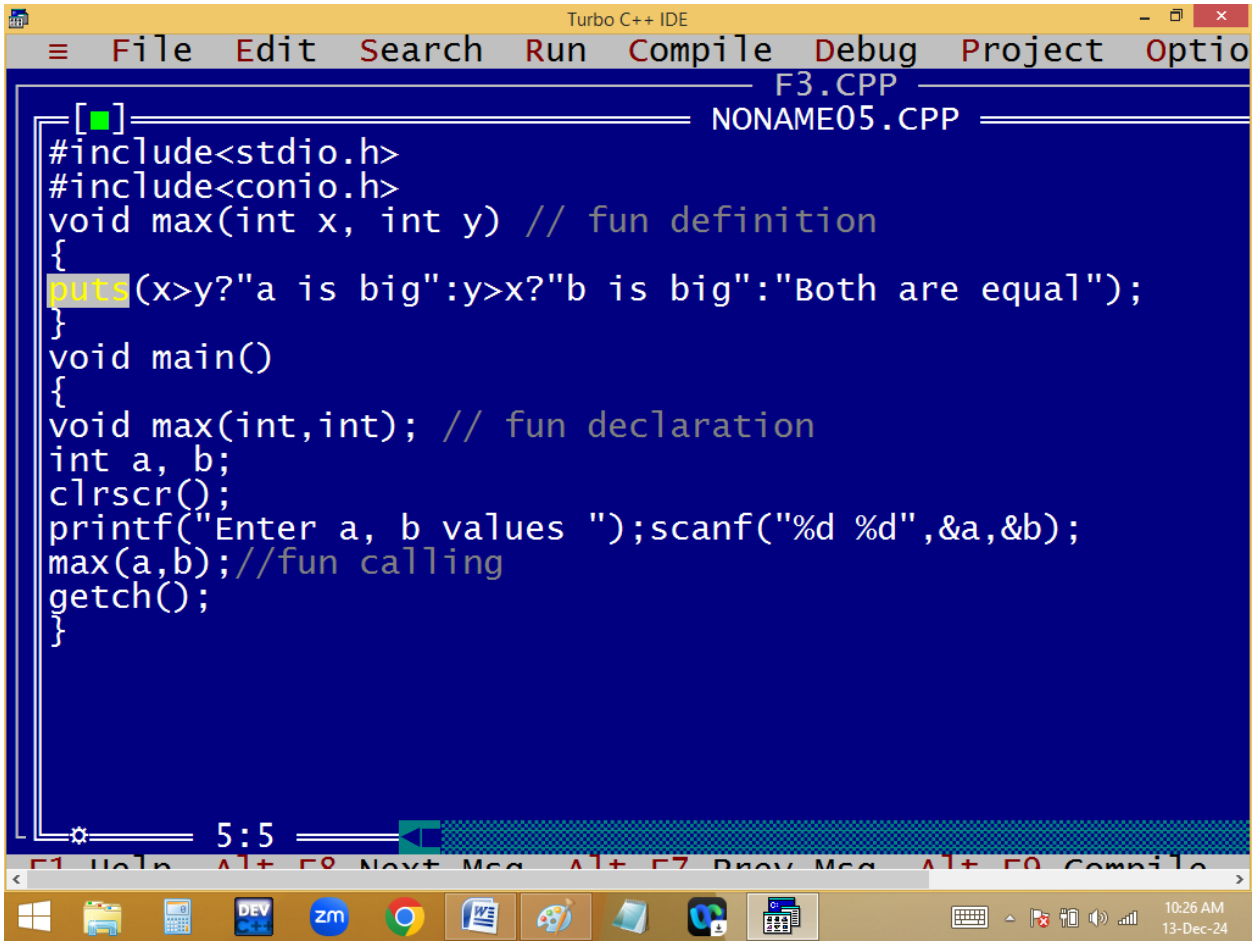

Turbo C++ IDE
Enter any no 7
Odd
odd
10:22 AM
13-Dec-24

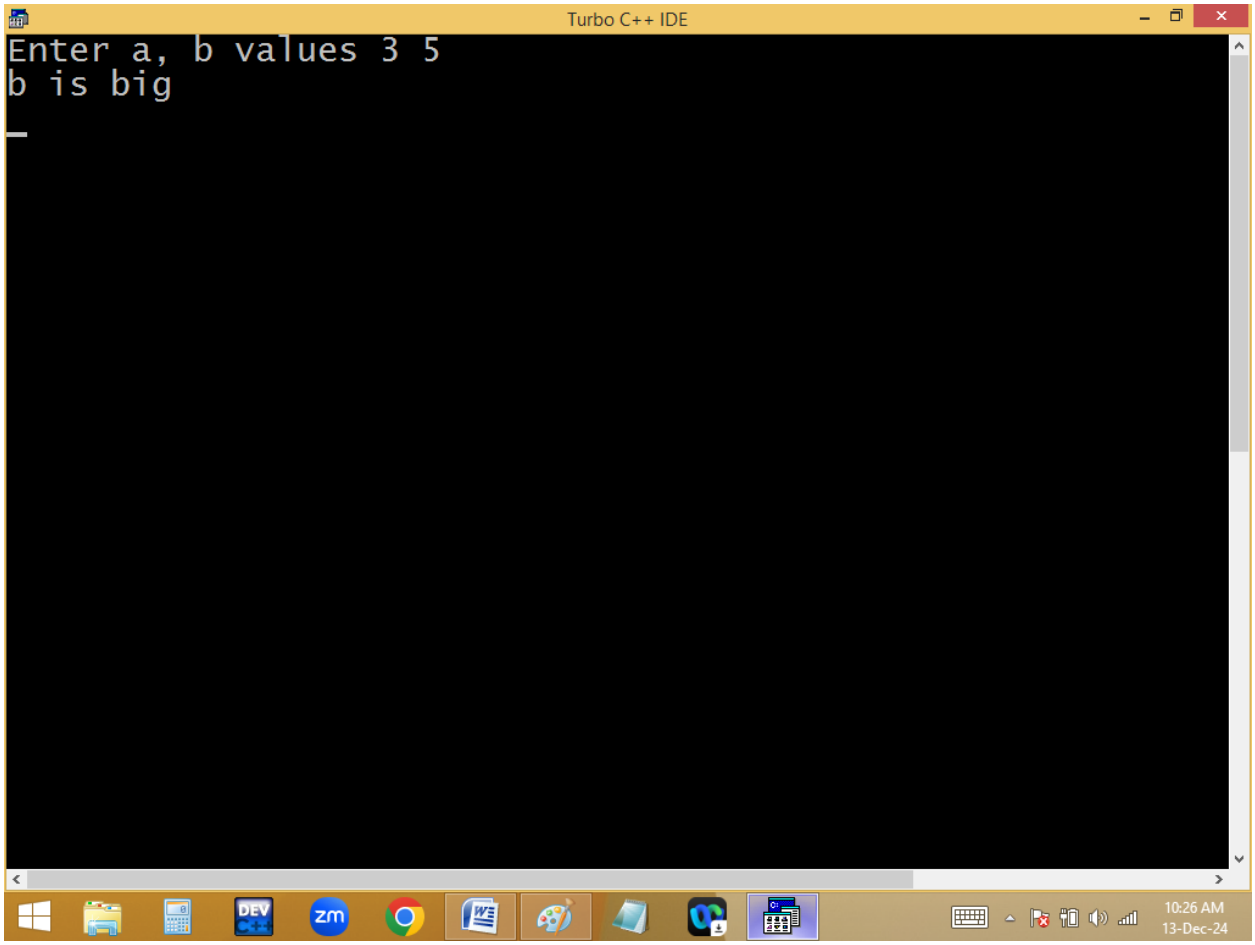


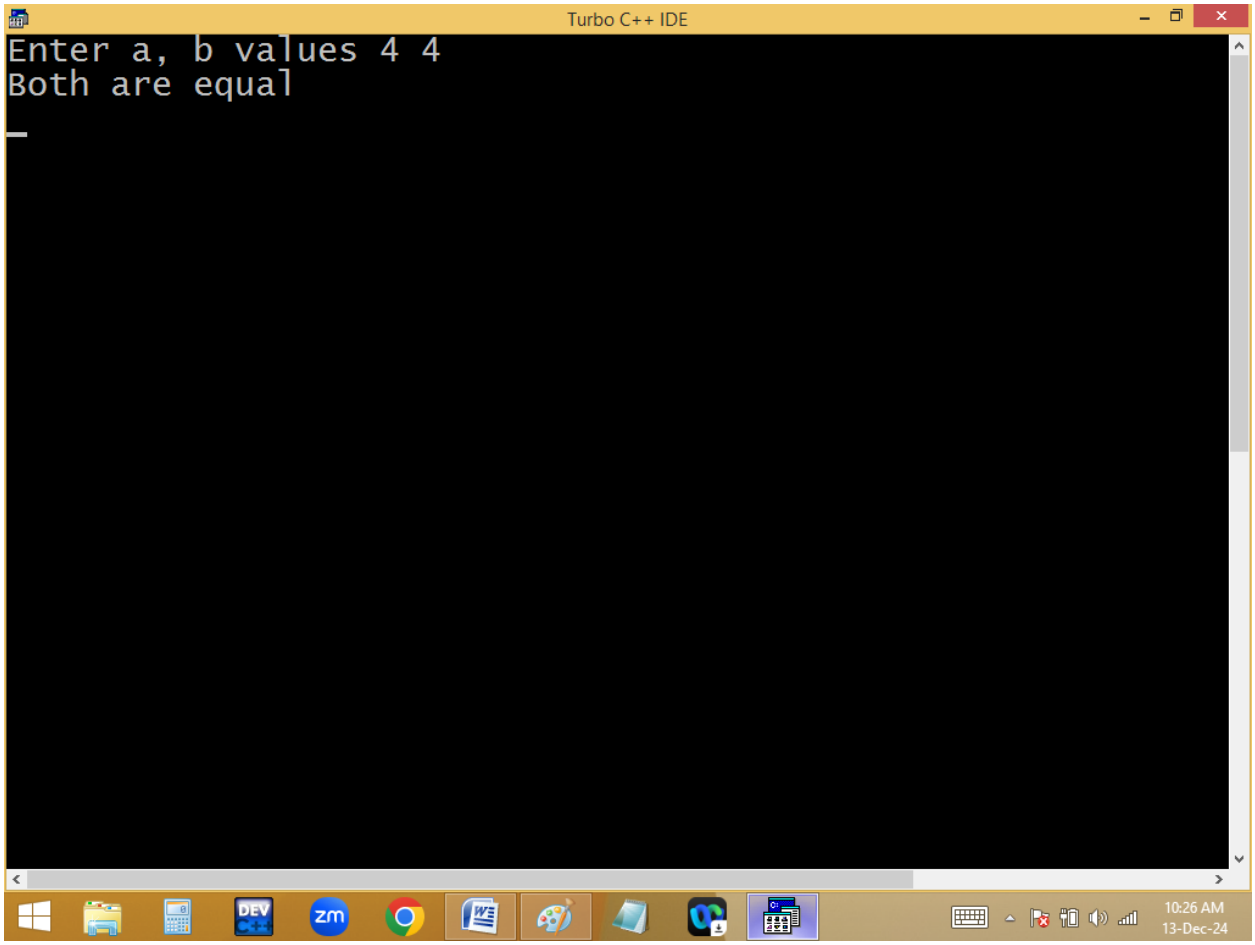
Function with arguments, without return value:

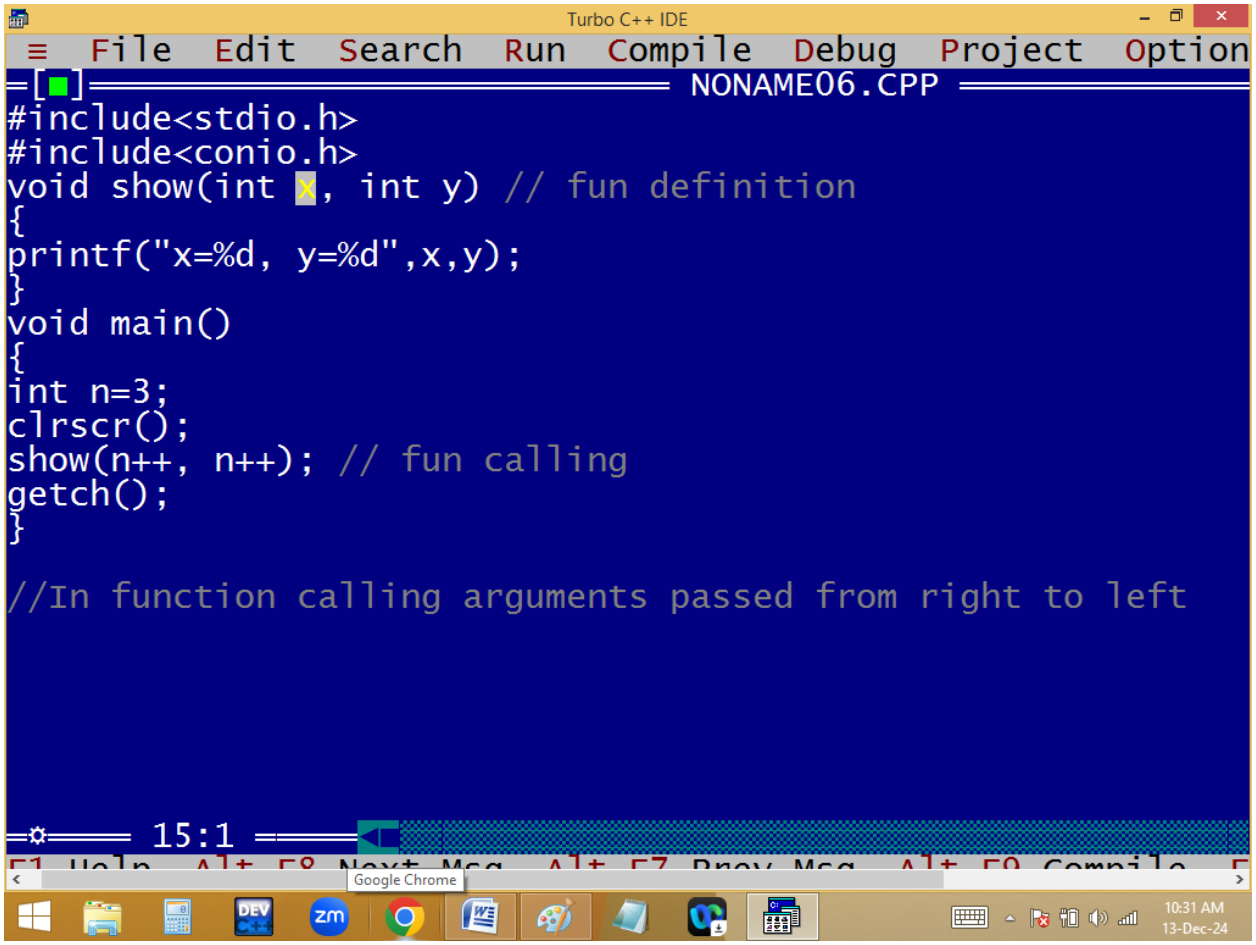

Turbo C++ IDE
File Edit Search Run Compile Debug Project Optio
F3.CPP
#include<stdio.h>
#include<conio.h>
void sum(int, int); // fun declaration
void main()
{
int a, b;
clrscr();
printf("Enter a, b values "); scanf("%d %d",&a,&b);
sum(a,b); // fun calling, a&b are actual parameters
getch();
}
void sum(int x, int y) // fun def, x&y formal parameters
{
printf("Sum=%d",x+y);
}
6:18
10:23 AM
13-Dec-24

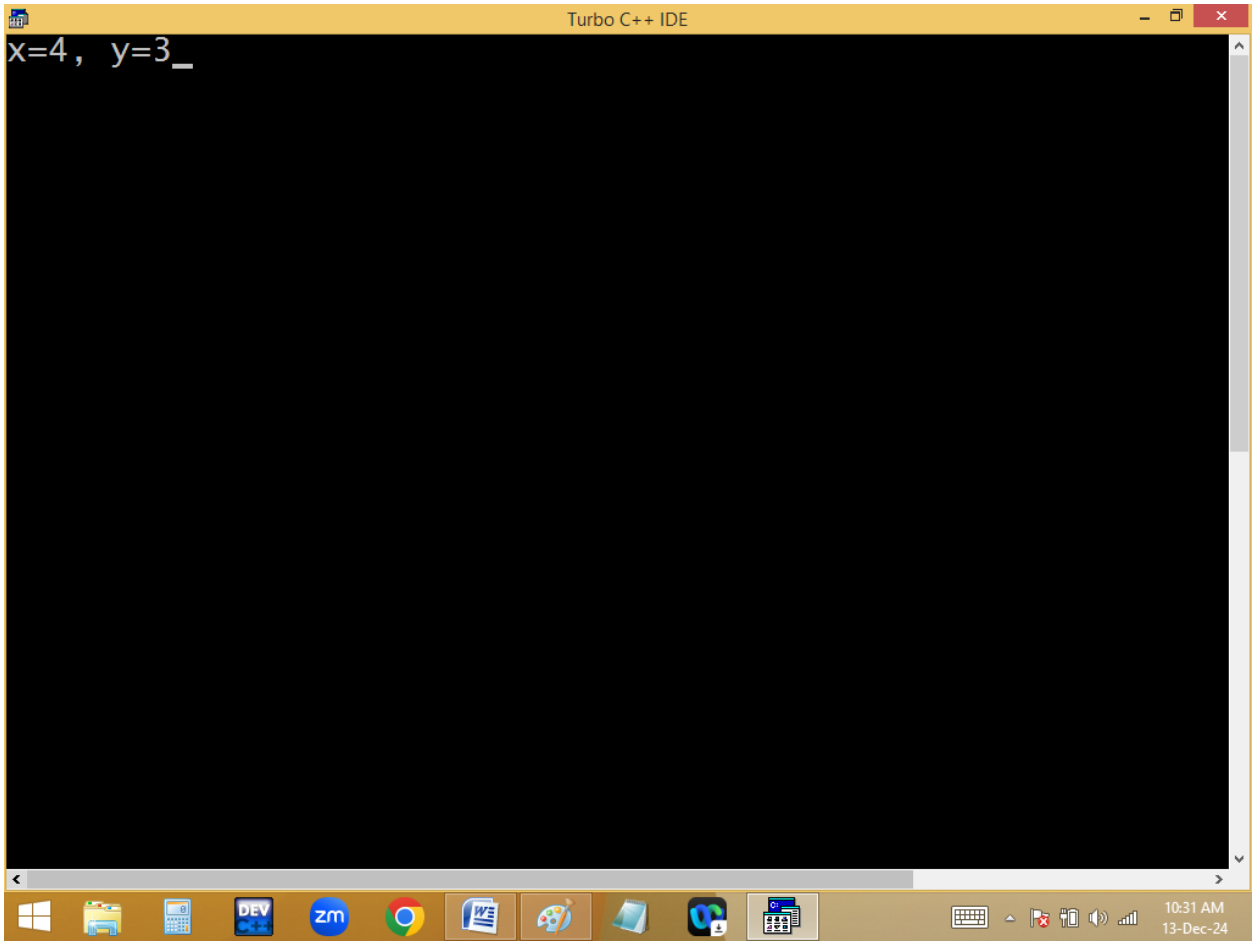

Turbo C++ IDE
Enter a, b values 2 3
Sum=5
10:23 AM
13-Dec-24


Turbo C++ IDE
File Edit Search Run Compile Debug Project Optio
F3.CPP
NONAME05.CPP
#include<stdio.h>
#include<conio.h>
void max(int x, int y) // fun definition
{
puts(x>y?"a is big":y>x?"b is big":"Both are equal");
}
void main()
{
void max(int,int); // fun declaration
int a, b;
clrscr();
printf("Enter a, b values ");scanf("%d %d",&a,&b);
max(a,b);//fun calling
getch();
}
5:5
10:26 AM
13-Dec-24

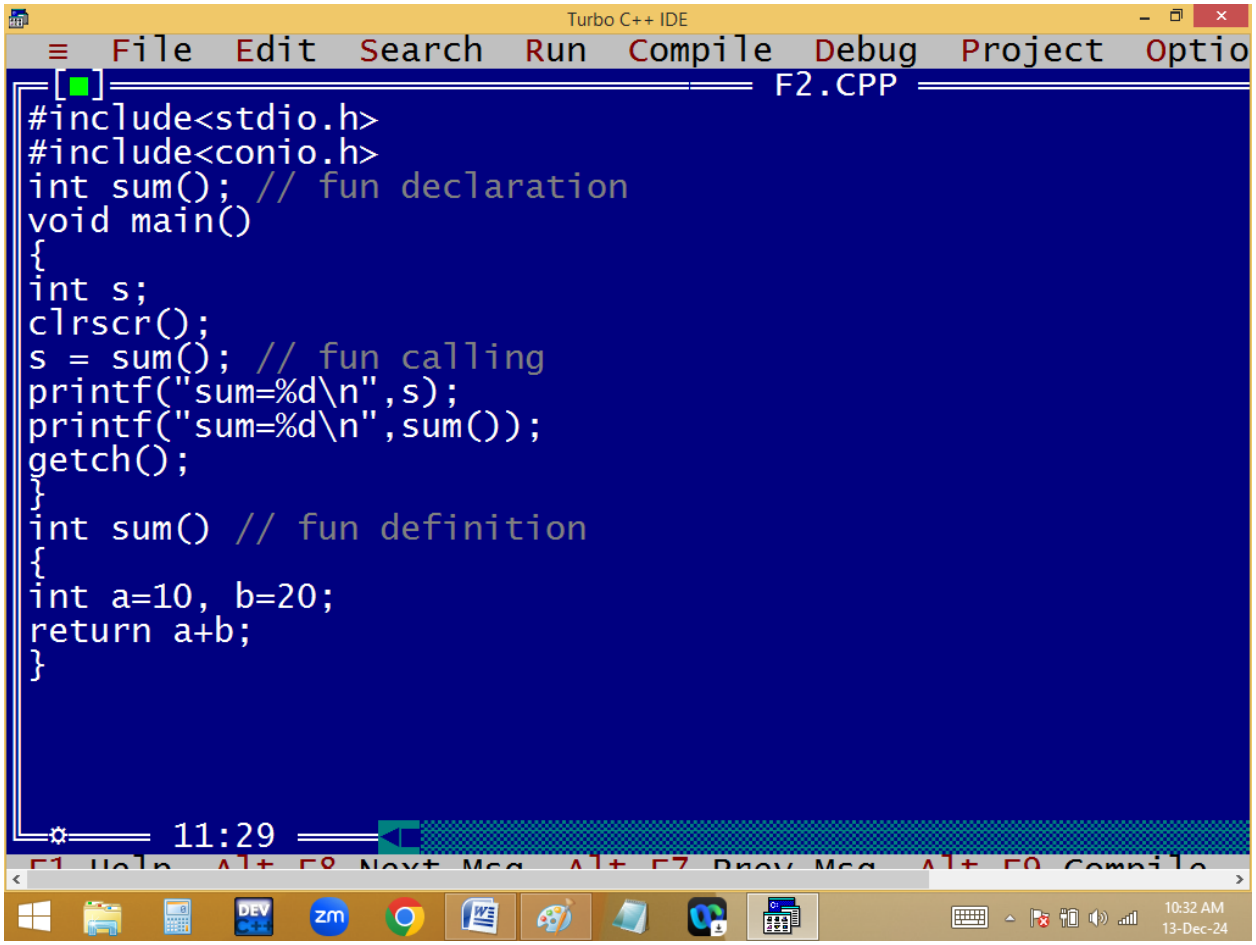

Turbo C++ IDE
Enter a, b values 3 5
b is big
10:26 AM
13-Dec-24

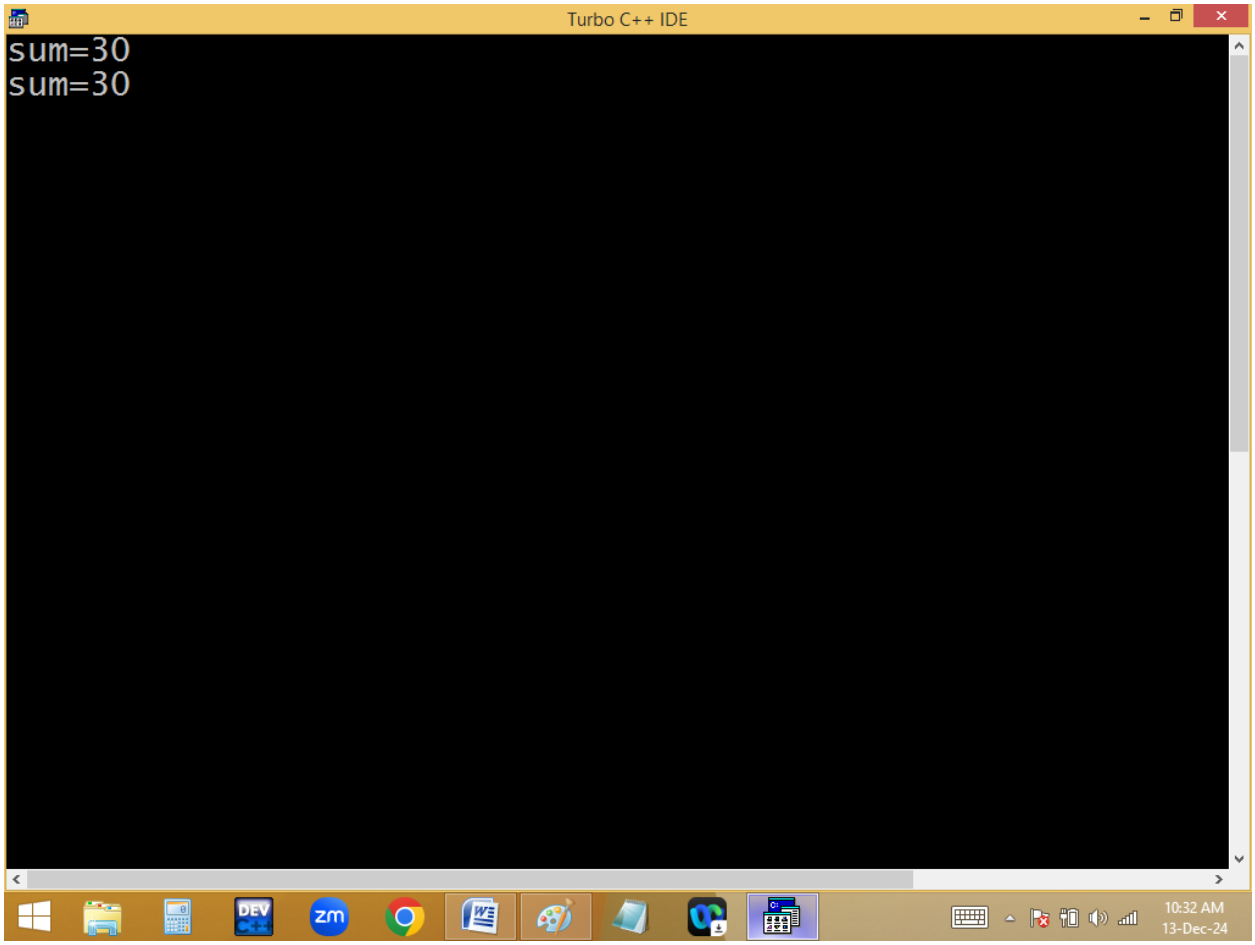

Turbo C++ IDE
Enter a, b values 4 4
Both are equal
10:26 AM
13-Dec-24

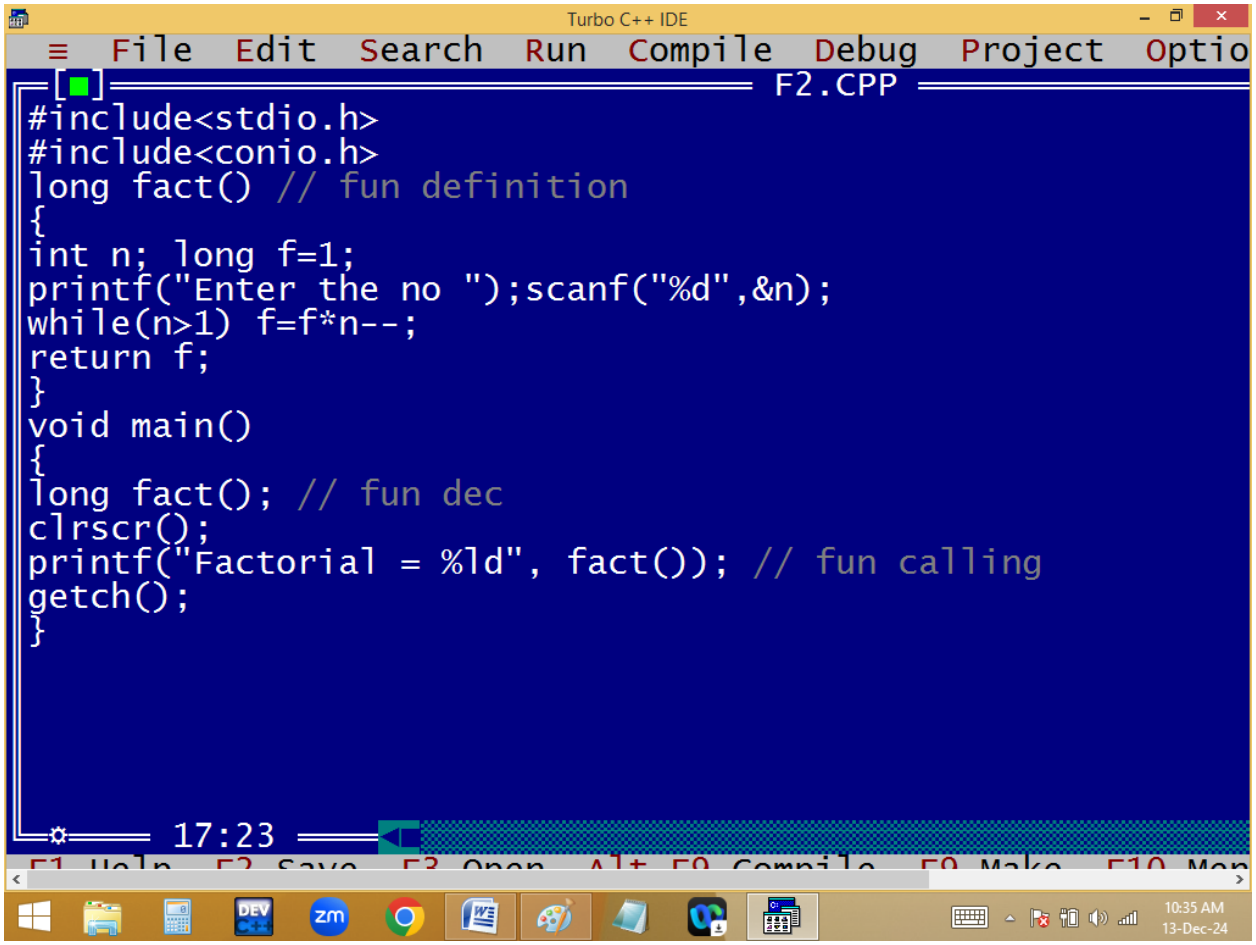

Turbo C++ IDE
File Edit Search Run Compile Debug Project Option
NONAME06.CPP
#include<stdio.h>
#include<conio.h>
void show(int x, int y) // fun definition
{
printf("x=%d, y=%d",x,y);
}
void main()
{
int n=3;
clrscr();
show(n++, n++); // fun calling
getch();
}
//In function calling arguments passed from right to left
15:1
Google Chrome
10:31 AM
13-Dec-24

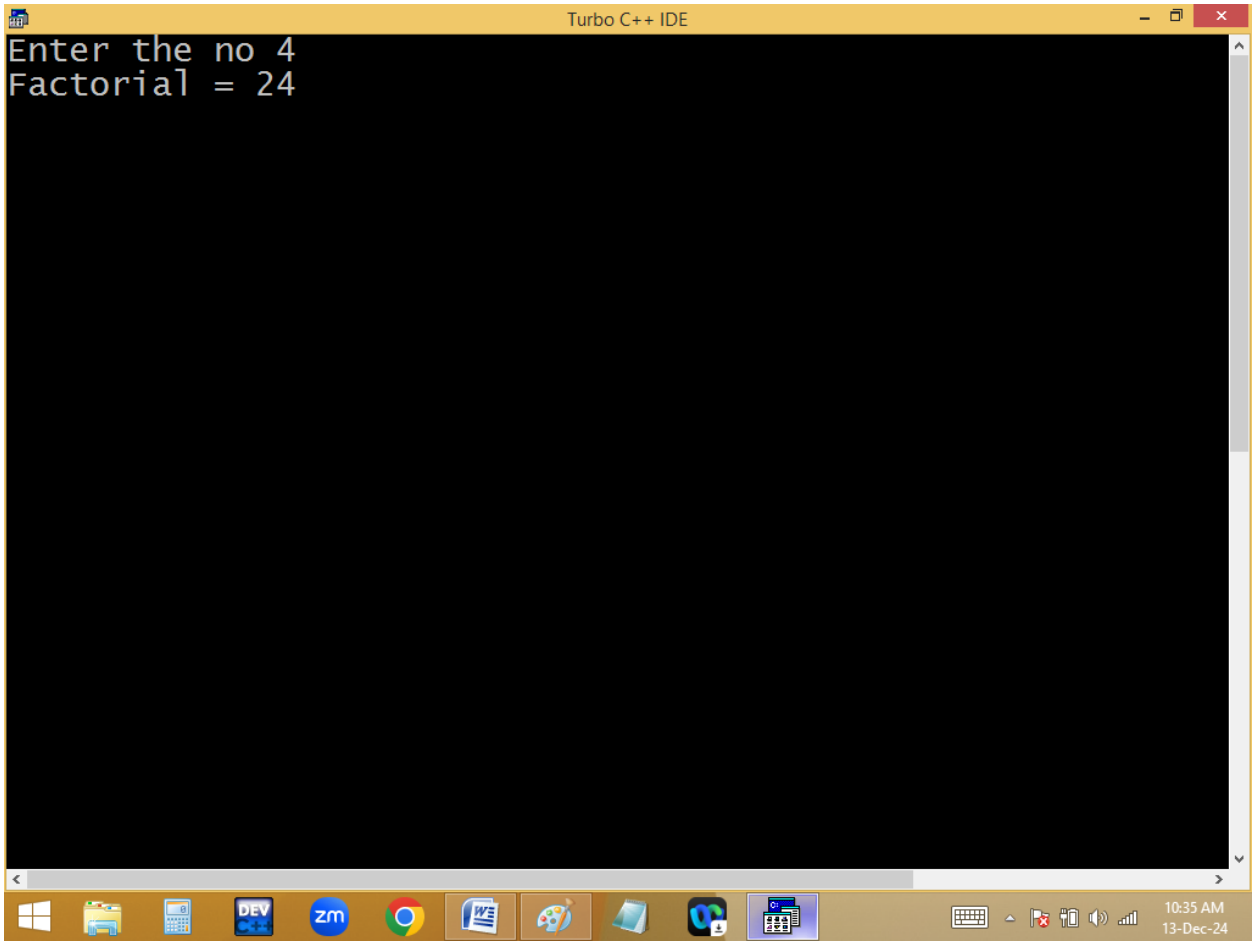


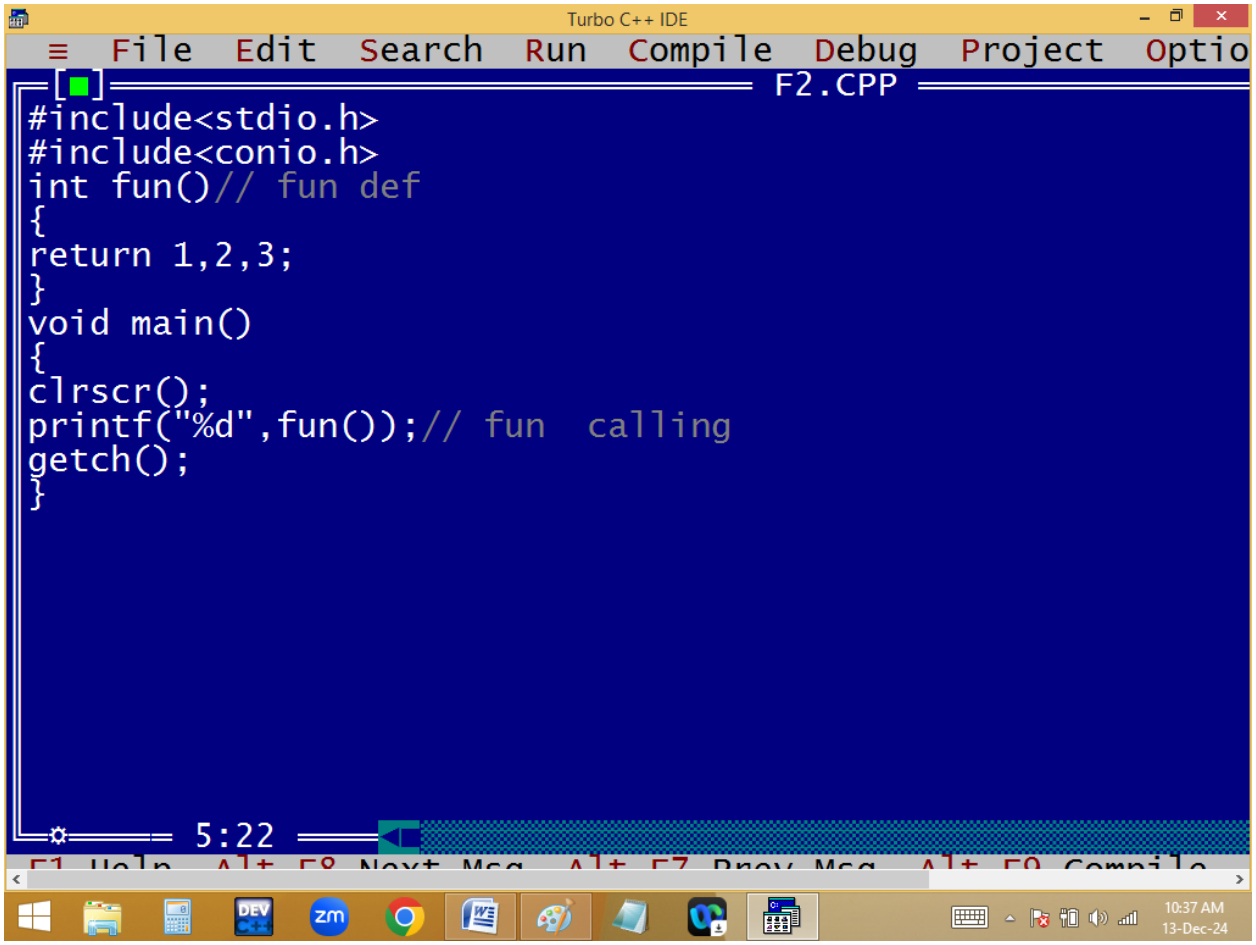
Function without arguments, with return value:

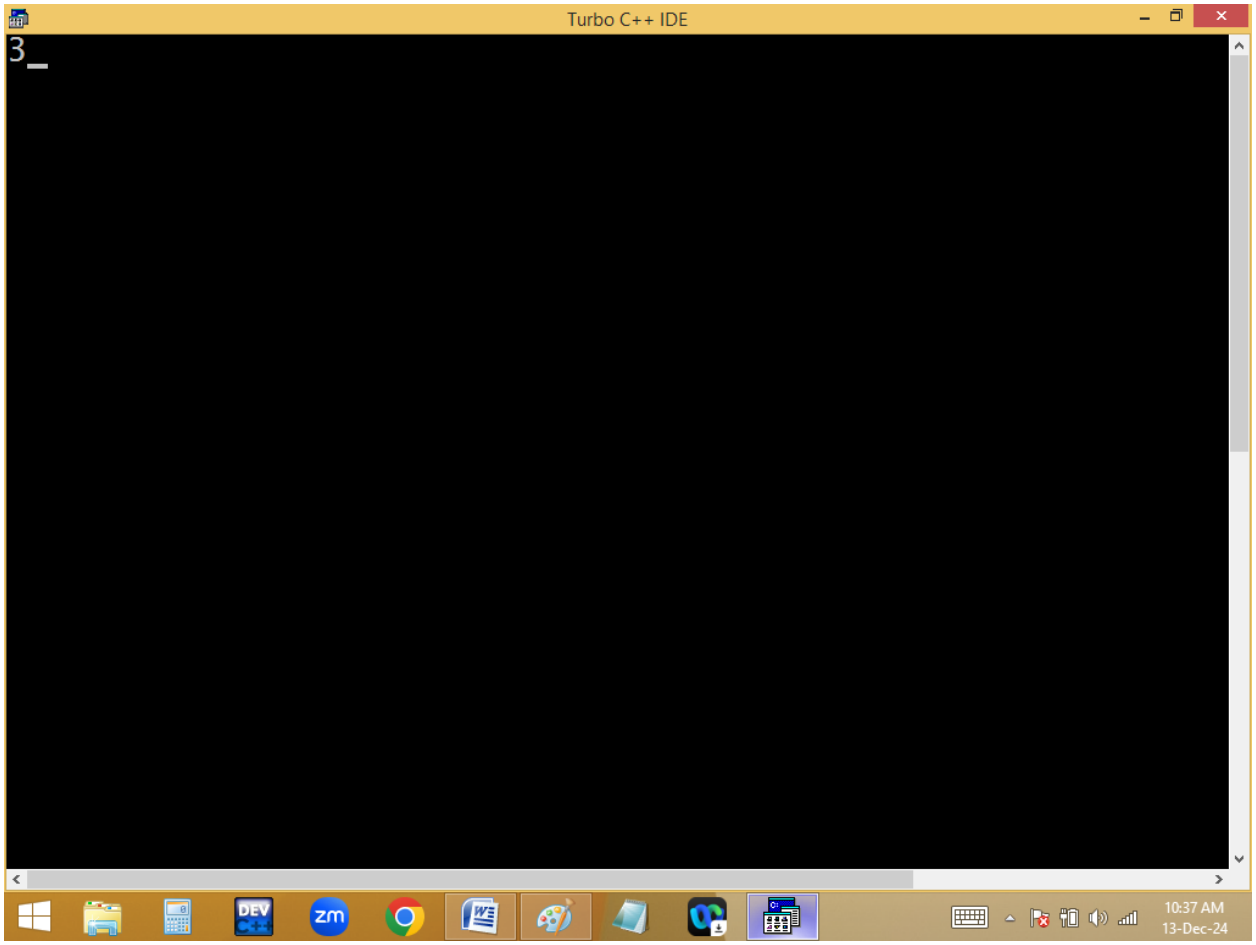

Turbo C++ IDE
File Edit Search Run Compile Debug Project Optio
F2.CPP
#include<stdio.h>
#include<conio.h>
int sum(); // fun declaration
void main()
{
int s;
clrscr();
s = sum(); // fun calling
printf("sum=%d\n",s);
printf("sum=%d\n",sum());
getch();
}
int sum() // fun definition
{
int a=10, b=20;
return a+b;
}
11:29
10:32 AM
13-Dec-24

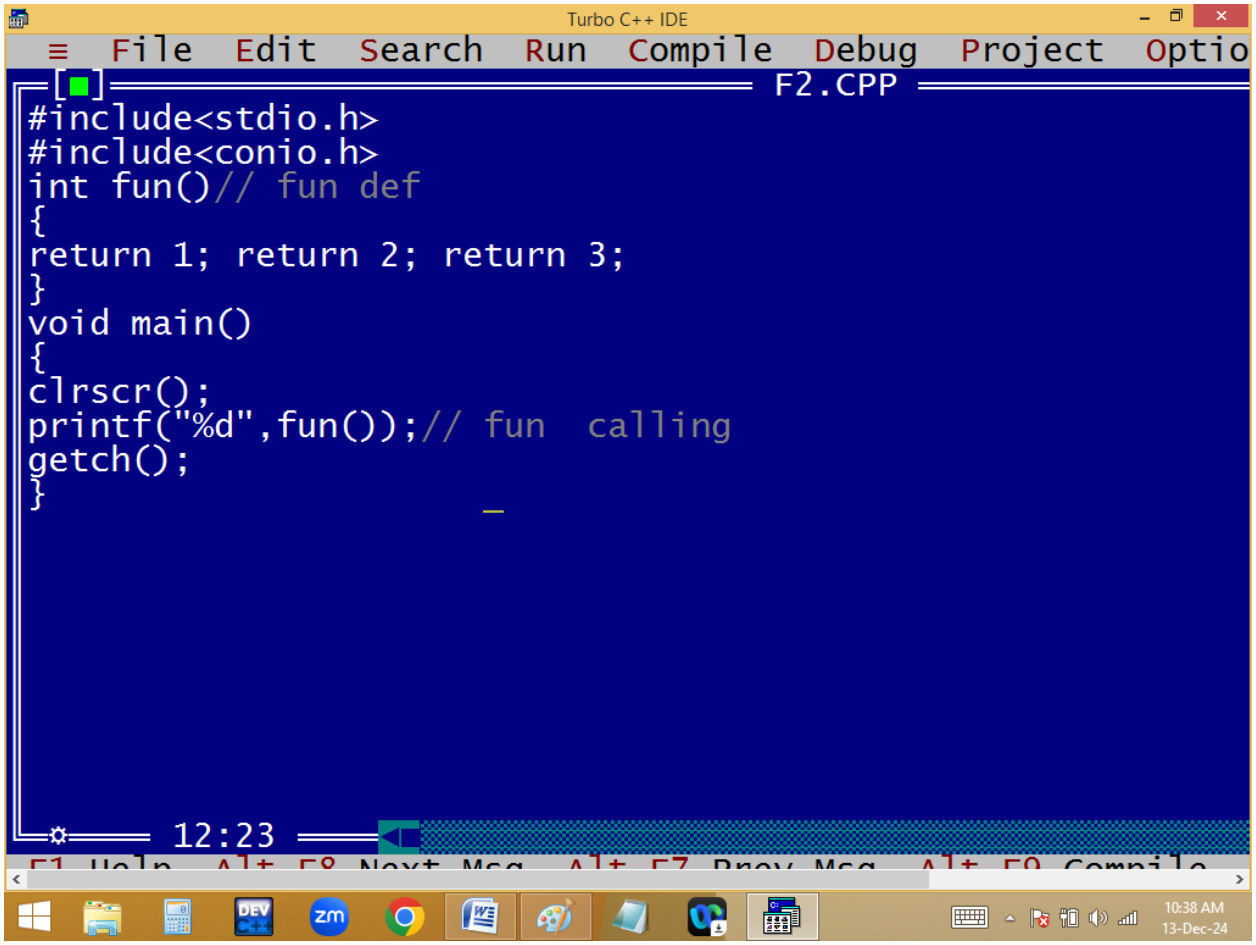

Turbo C++ IDE
sum=30
sum=30
10:32 AM
13-Dec-24

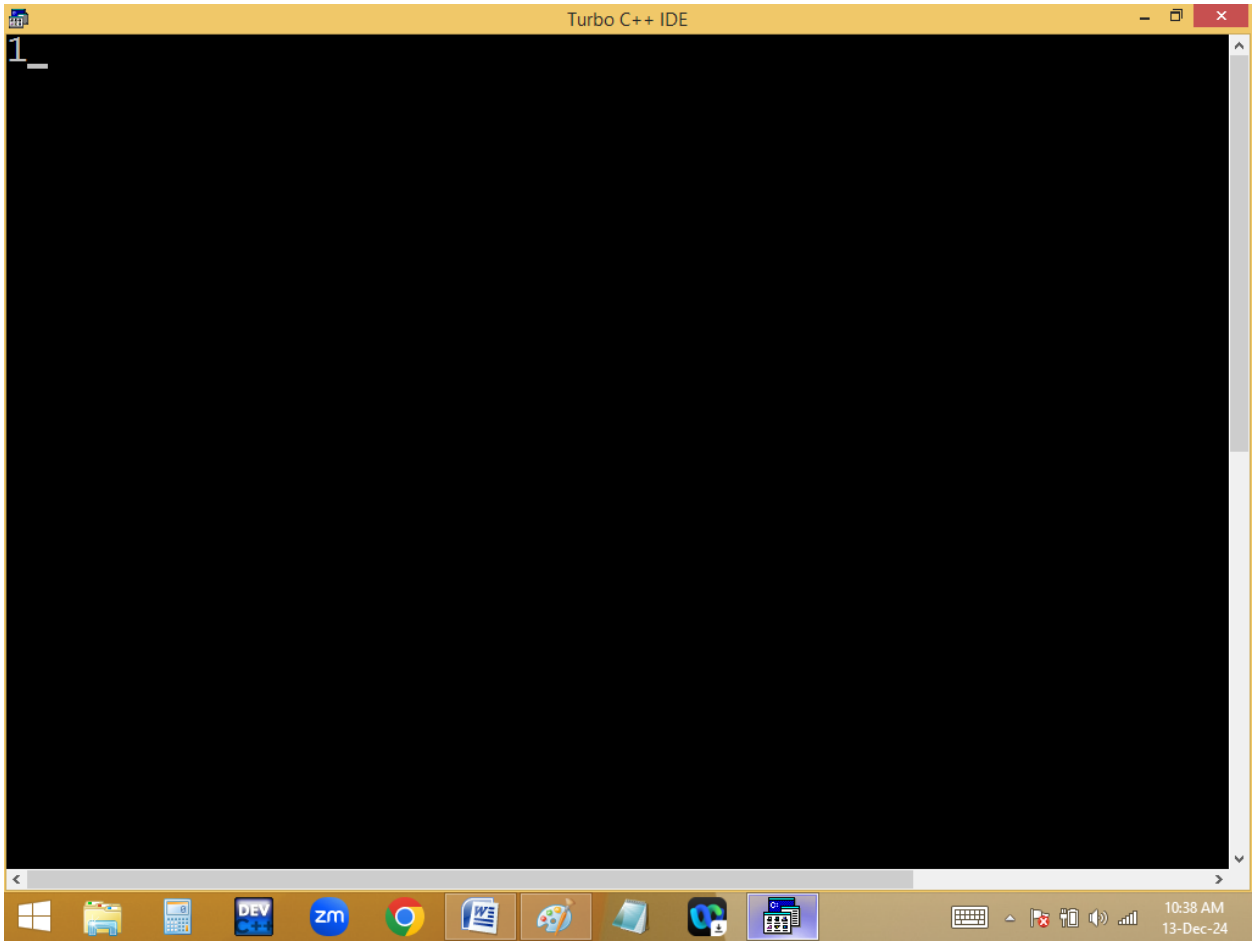

Turbo C++ IDE
File Edit Search Run Compile Debug Project Optio
F2.CPP
#include<stdio.h>
#include<conio.h>
long fact() // fun definition
{
int n; long f=1;
printf("Enter the no ");scanf("%d",&n);
while(n>1) f=f*n--;
return f;
}
void main()
{
long fact(); // fun dec
clrscr();
printf("Factorial = %ld", fact()); // fun calling
getch();
}
17:23
10:35 AM
13-Dec-24


Turbo C++ IDE
Enter the no 4
Factorial = 24
10:35 AM
13-Dec-24


Turbo C++ IDE
File Edit Search Run Compile Debug Project Optio
F2.CPP
#include<stdio.h>
#include<conio.h>
int fun()// fun def
{
return 1,2,3;
}
void main()
{
clrscr();
printf("%d",fun());// fun calling
getch();
}
5:22
10:37 AM
13-Dec-24

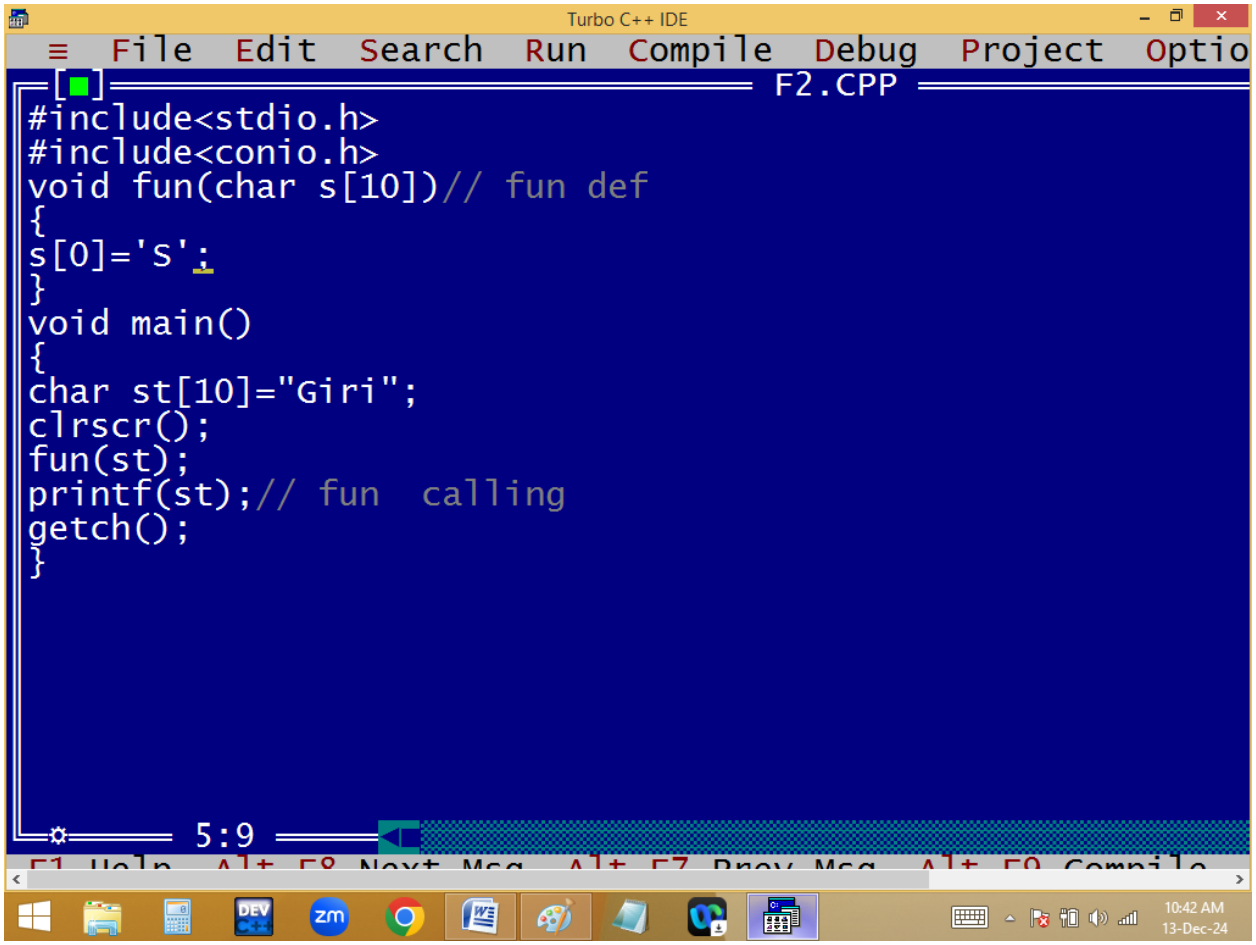

Turbo C++ IDE
3_
10:37 AM
13-Dec-24

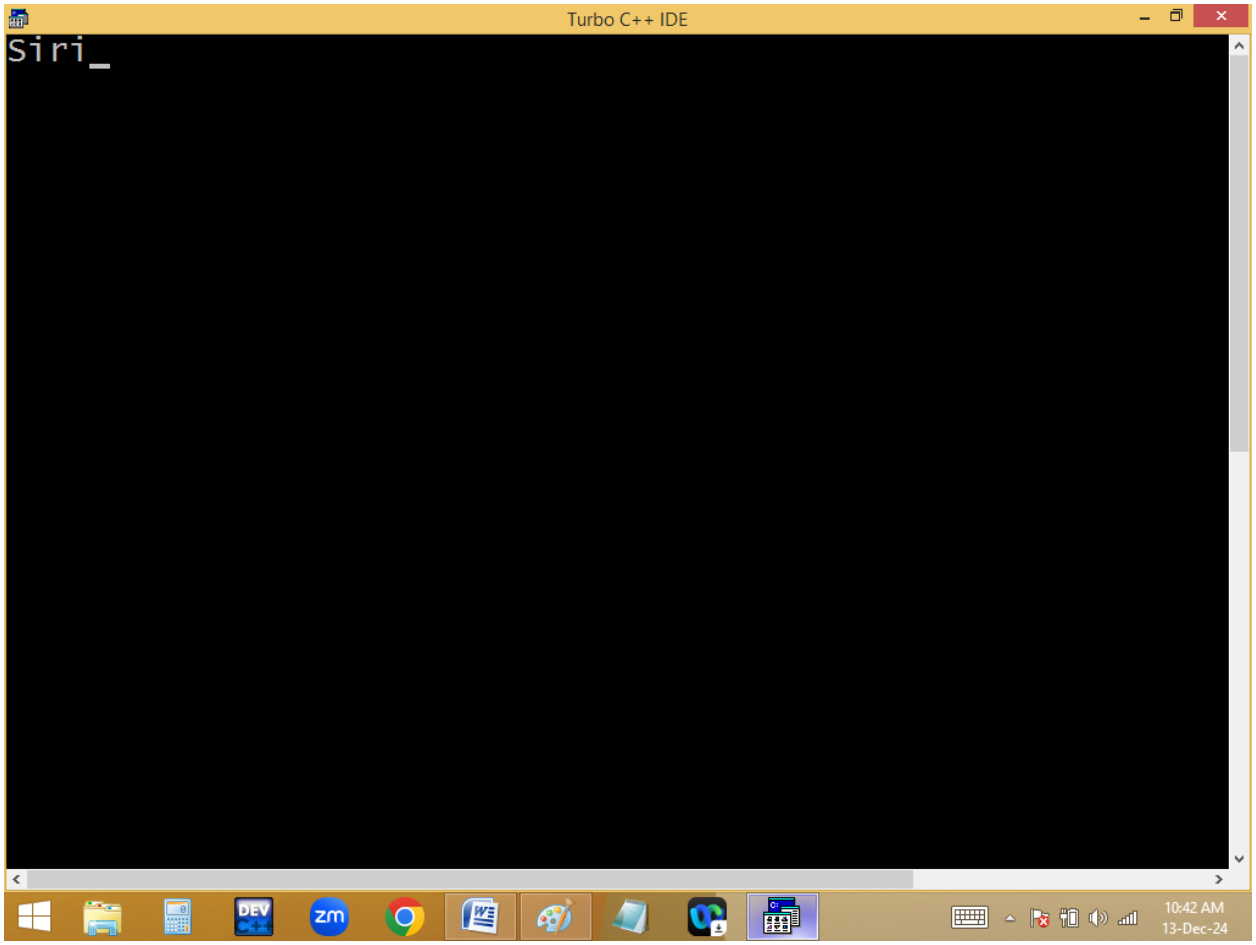

Turbo C++ IDE
File Edit Search Run Compile Debug Project Optio
F2.CPP
#include<stdio.h>
#include<conio.h>
int fun()// fun def
{
return 1; return 2; return 3;
}
void main()
{
clrscr();
printf("%d",fun());// fun calling
getch();
}
12:23
10:38 AM
13-Dec-24

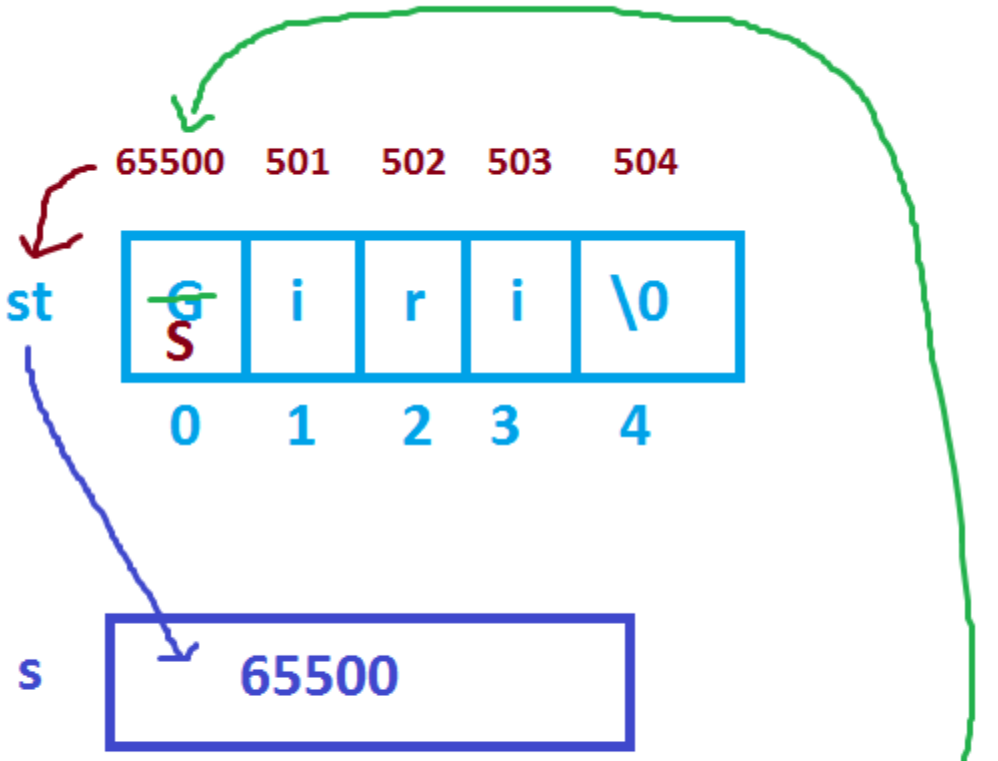


Passing array / string to function:

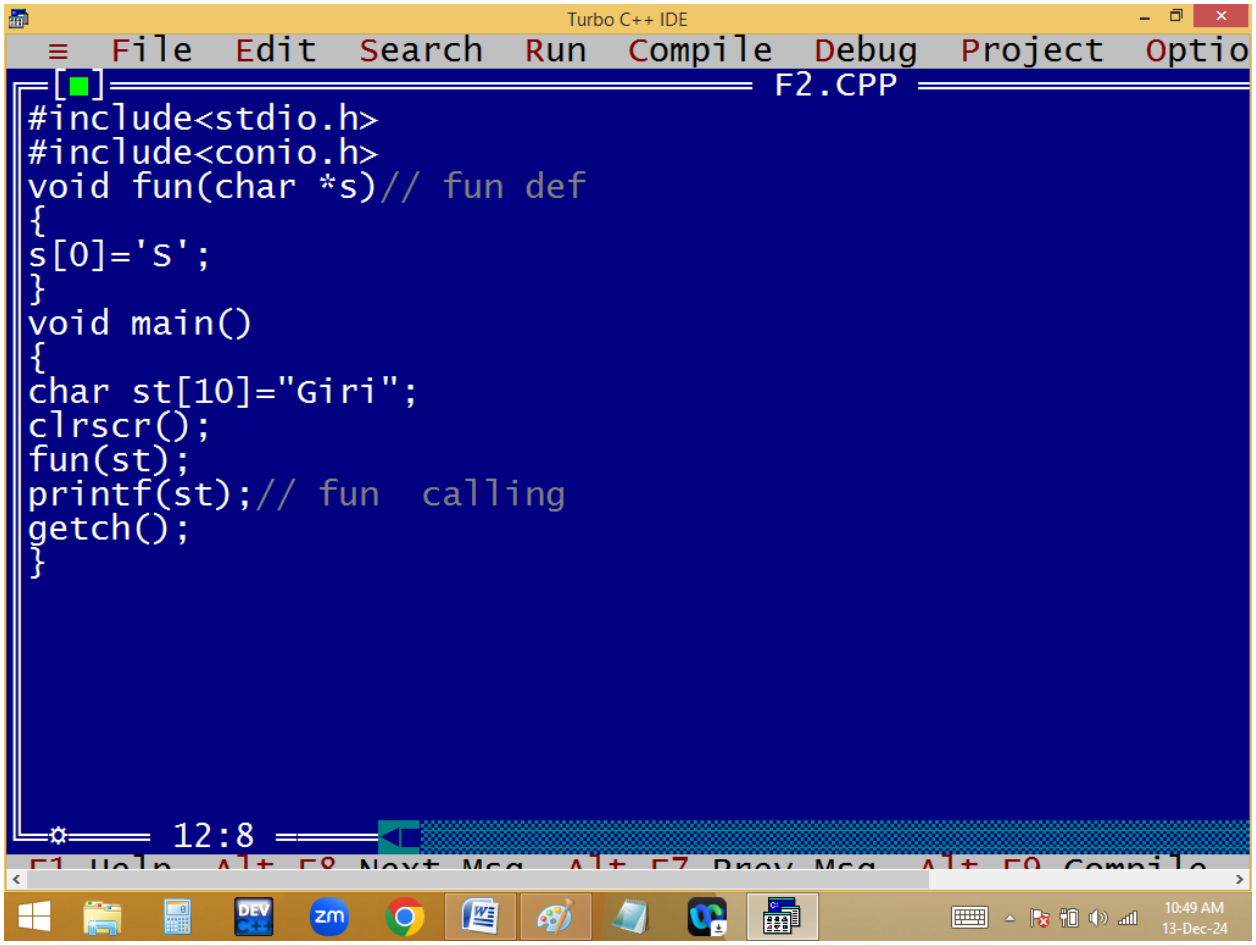
Call by reference / call by address:

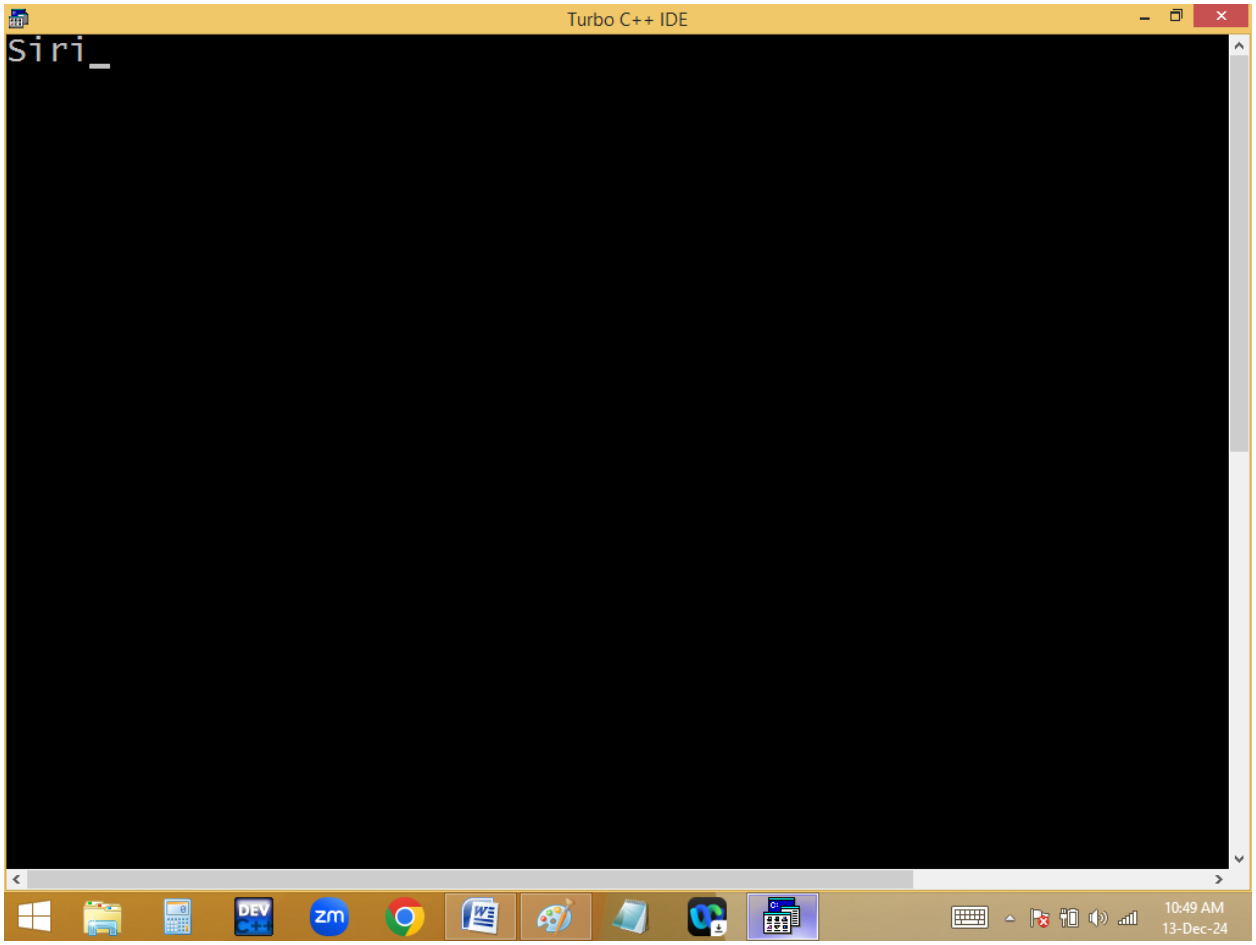

Turbo C++ IDE
File Edit Search Run Compile Debug Project Optio
F2.CPP
#include<stdio.h>
#include<conio.h>
void fun(char s[10])// fun def
{
s[0]='S';
}
void main()
{
char st[10]="Giri";
clrscr();
fun(st);
printf(st);// fun calling
getch();
}
5:9
10:42 AM
13-Dec-24

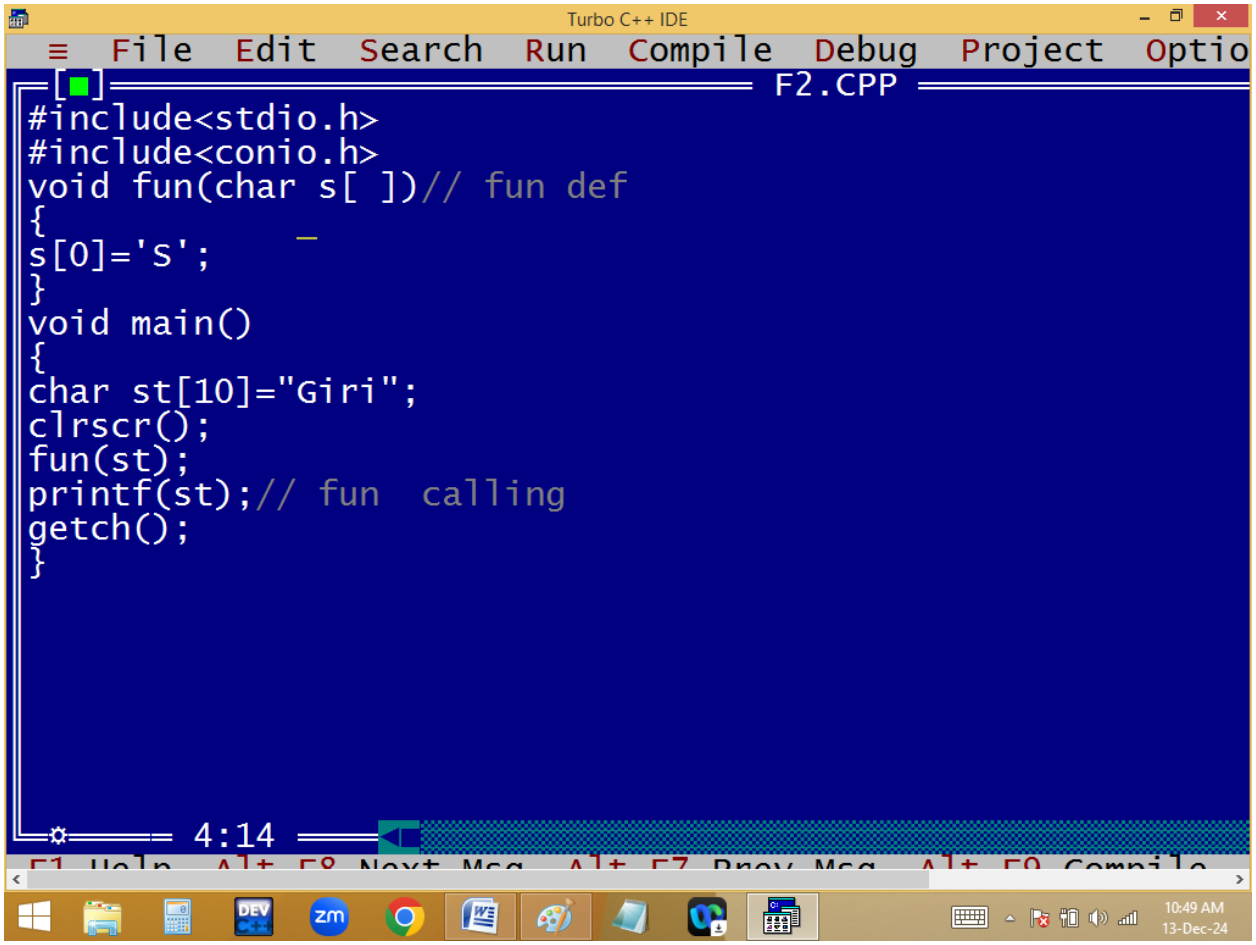

Turbo C++ IDE
Siri_
10:42 AM
13-Dec-24

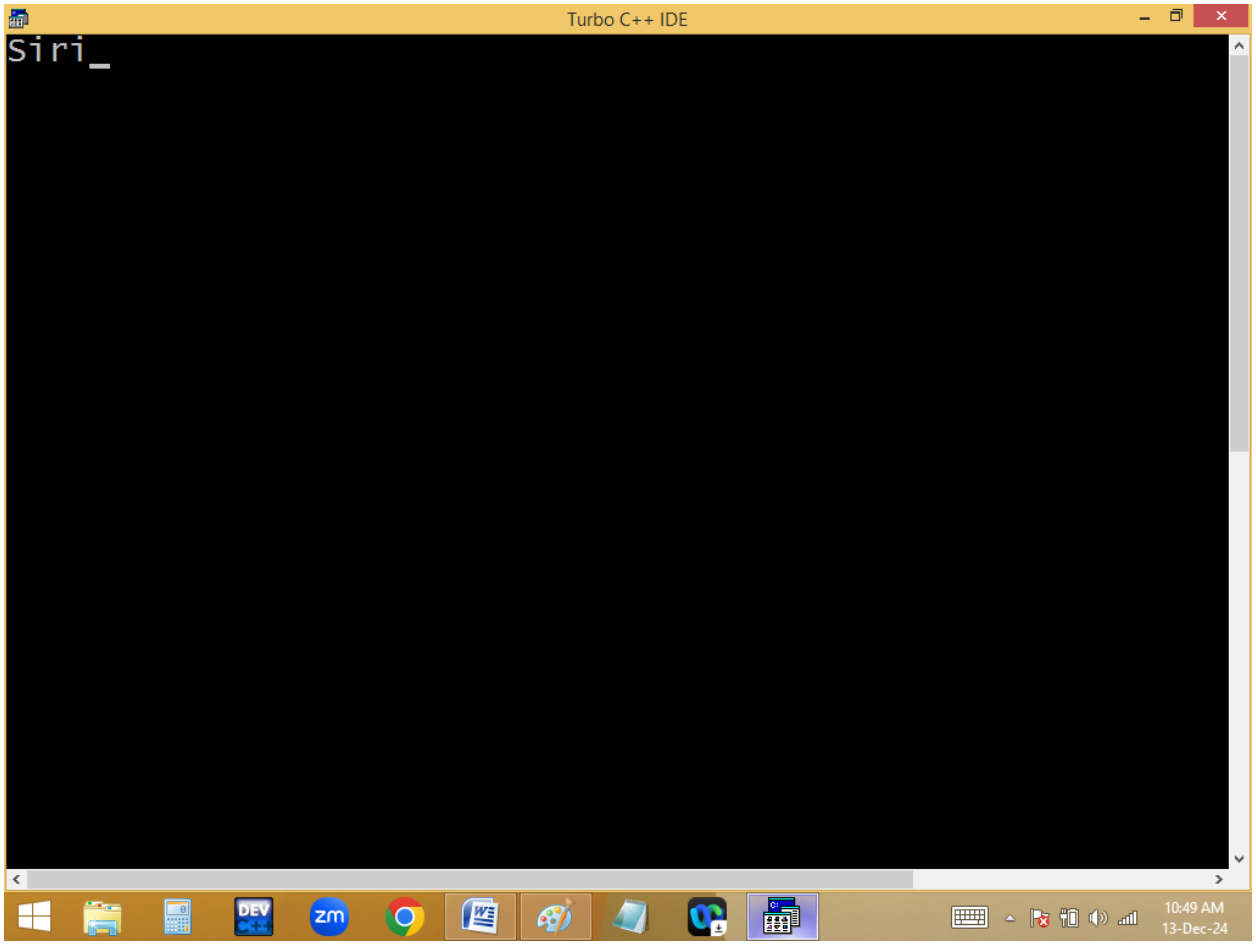


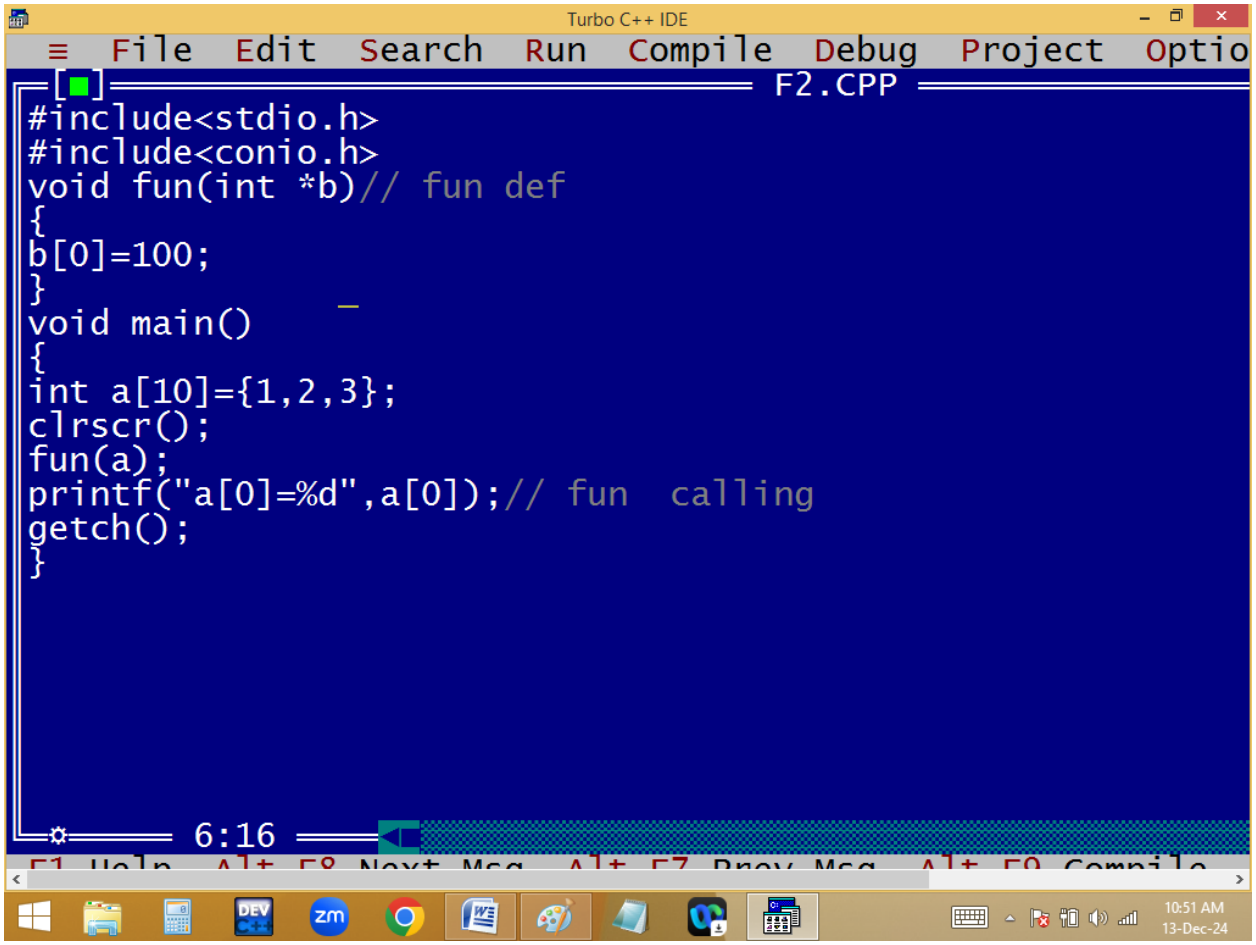
s[0]='G'; 65500+0*1=65500 = 'S';

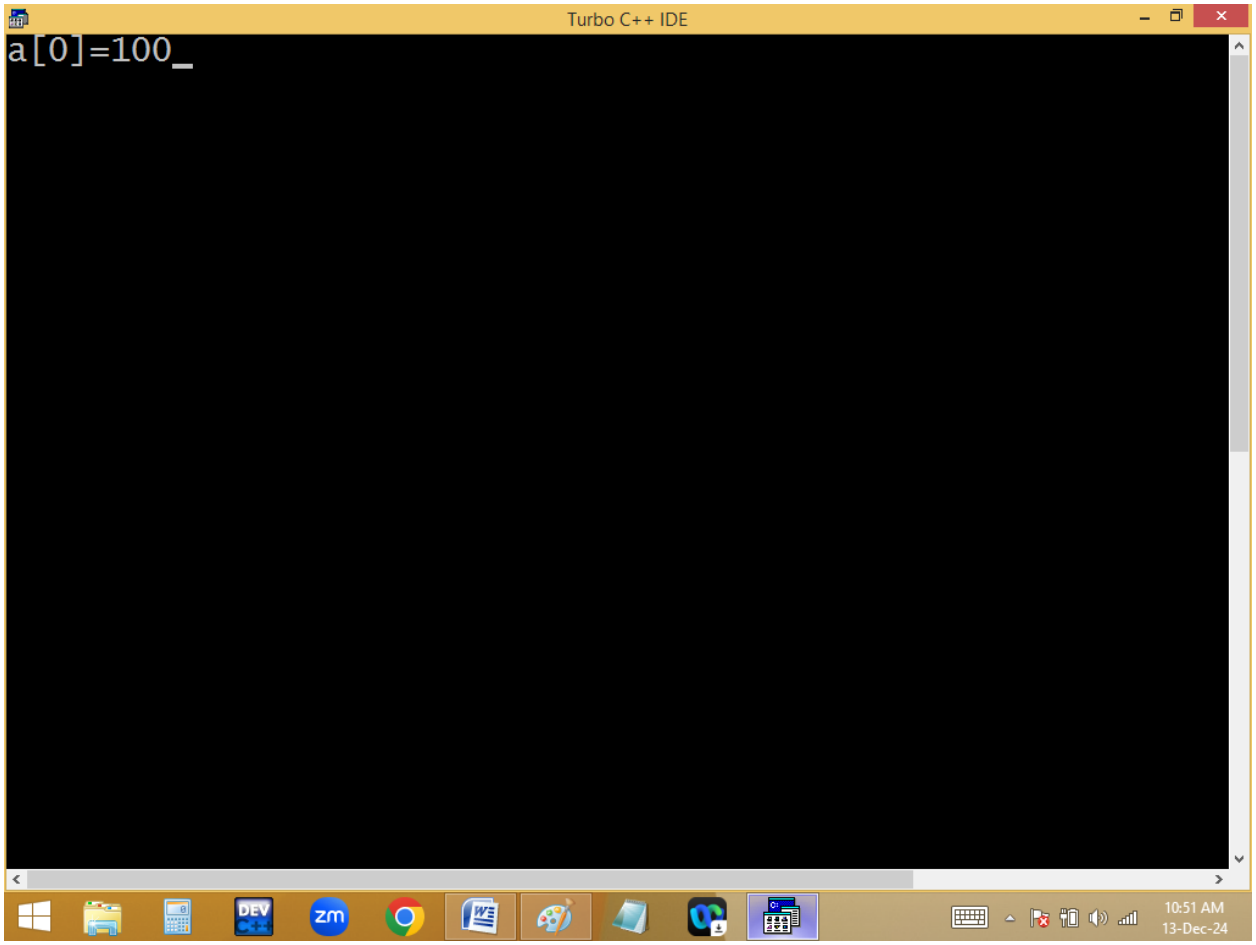

Turbo C++ IDE
File Edit Search Run Compile Debug Project Optio
F2.CPP
#include<stdio.h>
#include<conio.h>
void fun(char *s)// fun def
{
s[0]='S';
}
void main()
{
char st[10]="Giri";
clrscr();
fun(st);
printf(st);// fun calling
getch();
}
12:8
10:49 AM
13-Dec-24


Turbo C++ IDE
Siri_
10:49 AM
13-Dec-24


Turbo C++ IDE
File Edit Search Run Compile Debug Project Optio
F2.CPP
#include<stdio.h>
#include<conio.h>
void fun(char s[ ])// fun def
{
s[0]='S';
}
void main()
{
char st[10]="Giri";
clrscr();
fun(st);
printf(st);// fun calling
getch();
}
4:14
10:49 AM
13-Dec-24


Turbo C++ IDE
Siri_
10:49 AM
13-Dec-24


Turbo C++ IDE
File Edit Search Run Compile Debug Project Optio
F2.CPP
#include<stdio.h>
#include<conio.h>
void fun(int *b)// fun def
{
b[0]=100;
}
void main()
{
int a[10]={1,2,3};
clrscr();
fun(a);
printf("a[0]=%d",a[0]);// fun calling
getch();
}
6:16
10:51 AM
13-Dec-24


Turbo C++ IDE
a[0]=100_
10:51 AM
13-Dec-24

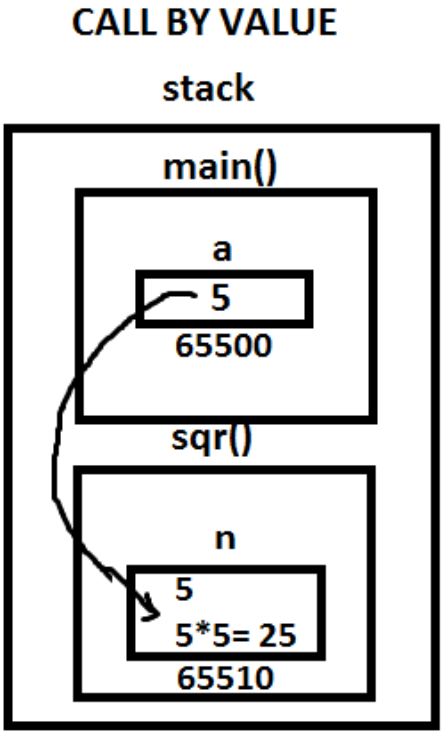
Passing parameters to the functions: [parameter passing techniques]

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;
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

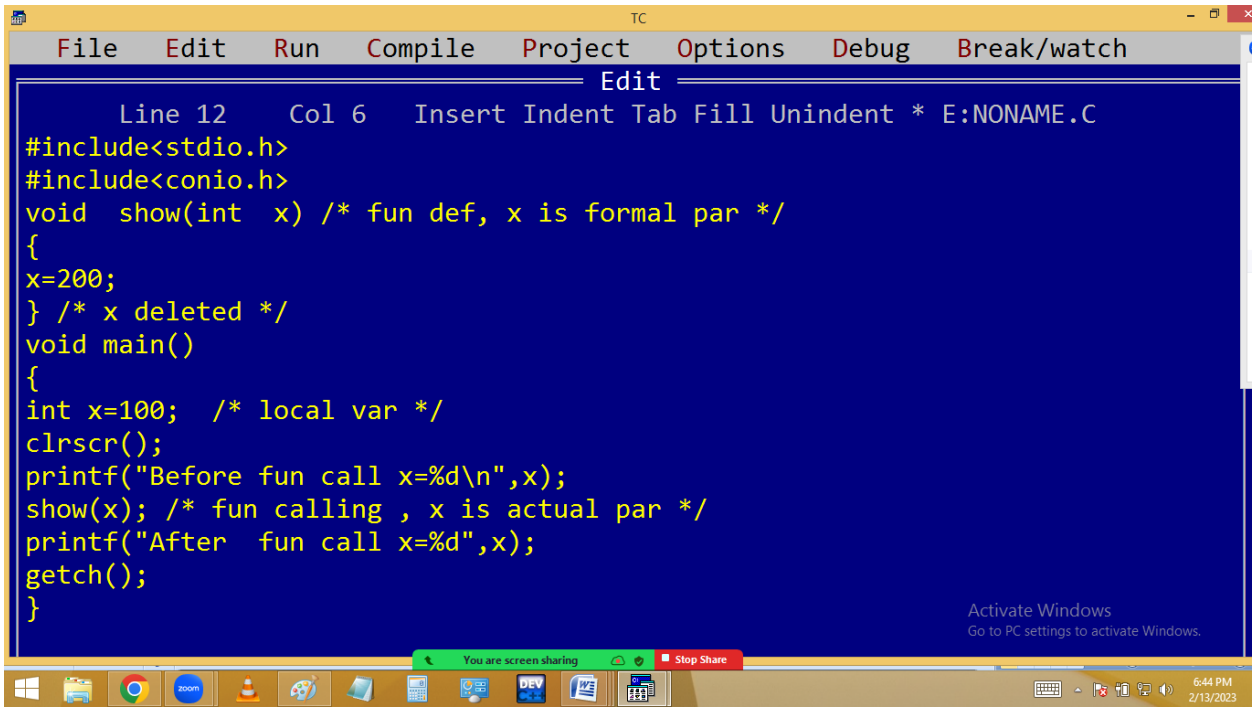
Eg: 2 swapping of two integers

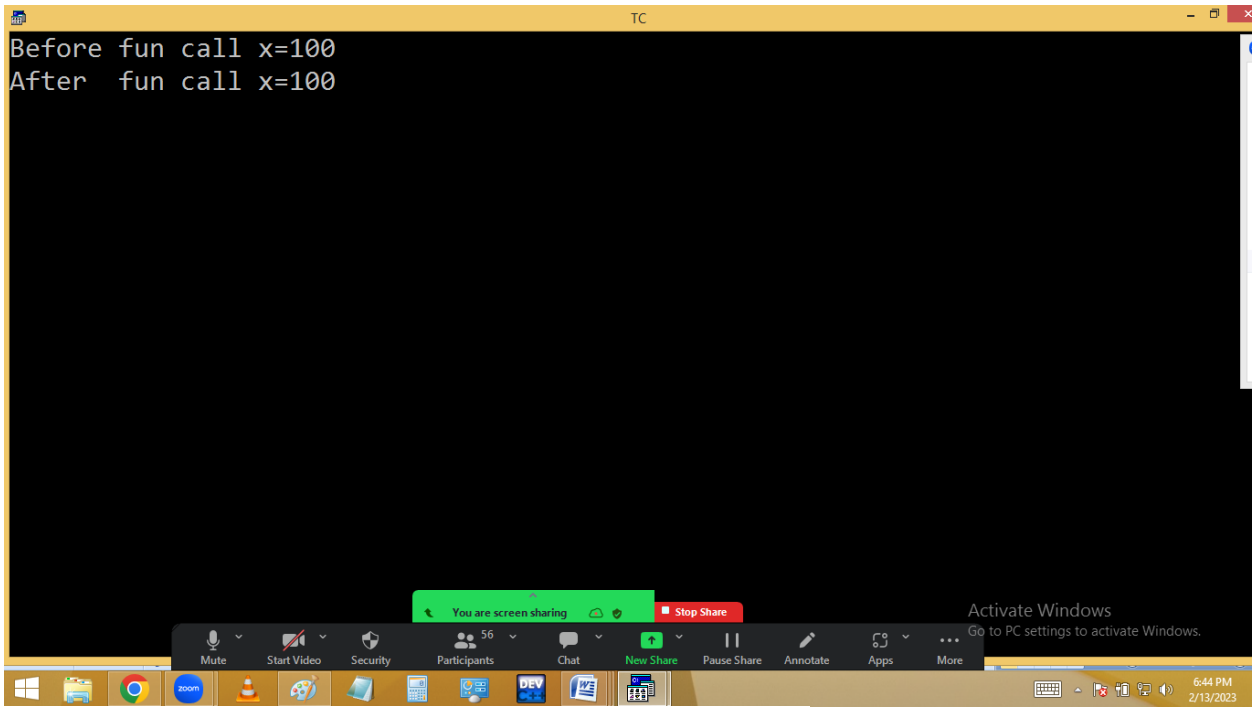
```
#include<stdio.h>
#include<conio.h>
void  swap(int  a, int  b)
{
int  temp=a;
a=b;
b=temp;
}
```

```
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();
}
```

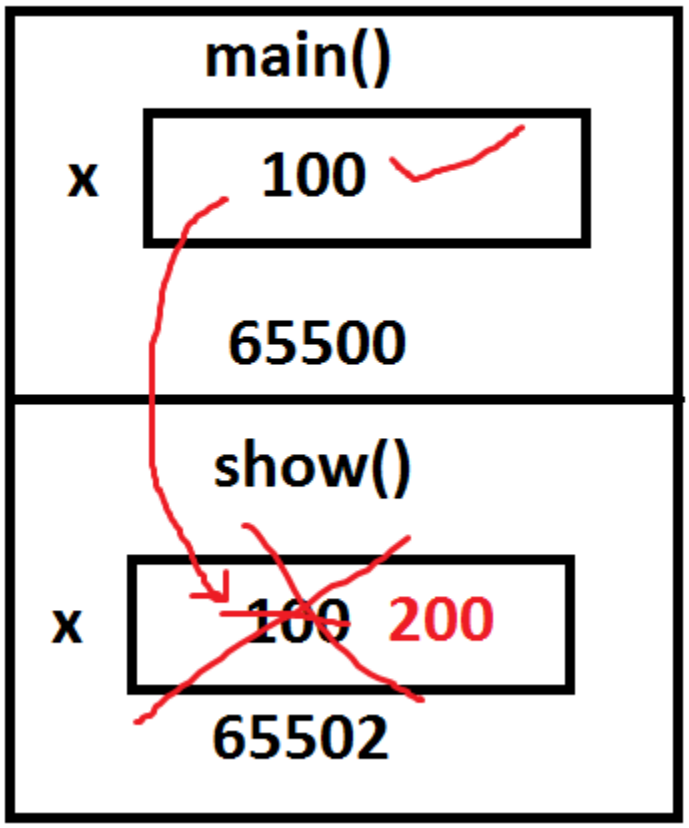
Output:

```
Before fun call a=5, b=7
After fun call a=5, b=7
```


TC
File Edit Run Compile Project Options Debug Break/watch
Edit
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();
}
Activate Windows
Go to PC settings to activate Windows.
You are screen sharing
Stop Share
6:44 PM
2/13/2023



Call by /pass By value



Call by address [Reference]:

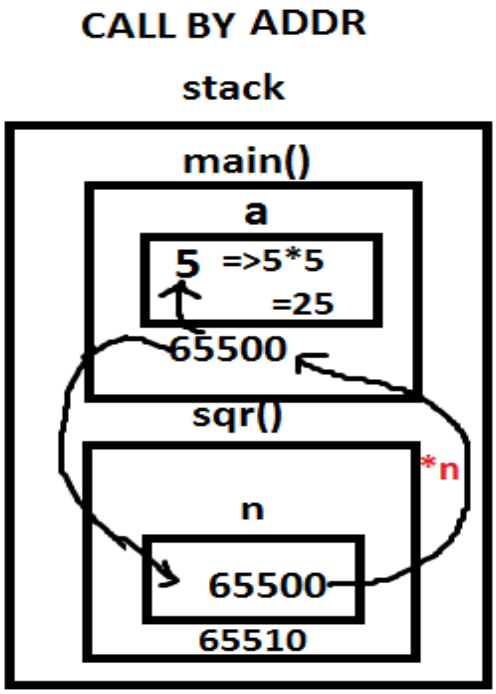
In call by address, the address of actual parameter is passed to formal parameter. Due to this the formal parameter should be declared as a pointer. Then only the formal parameter receives the actual parameter address. Due to this any changes in formal parameter effects in actual parameter address i.e.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

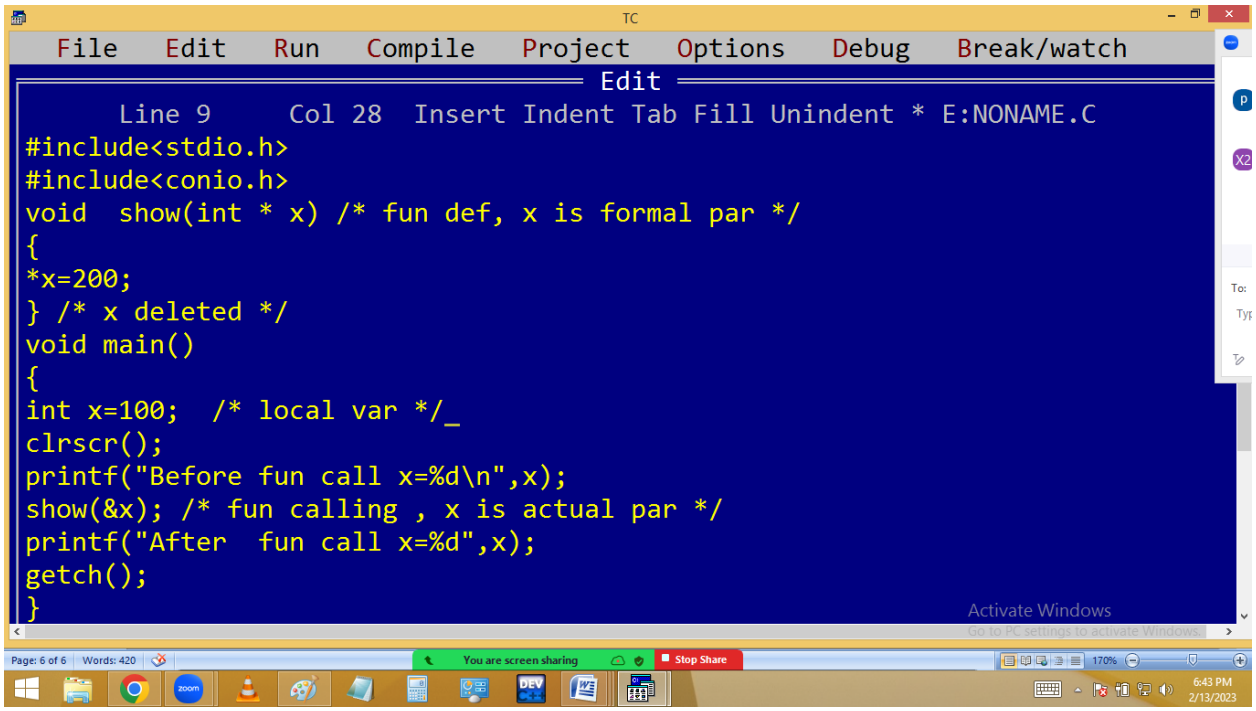
getch();

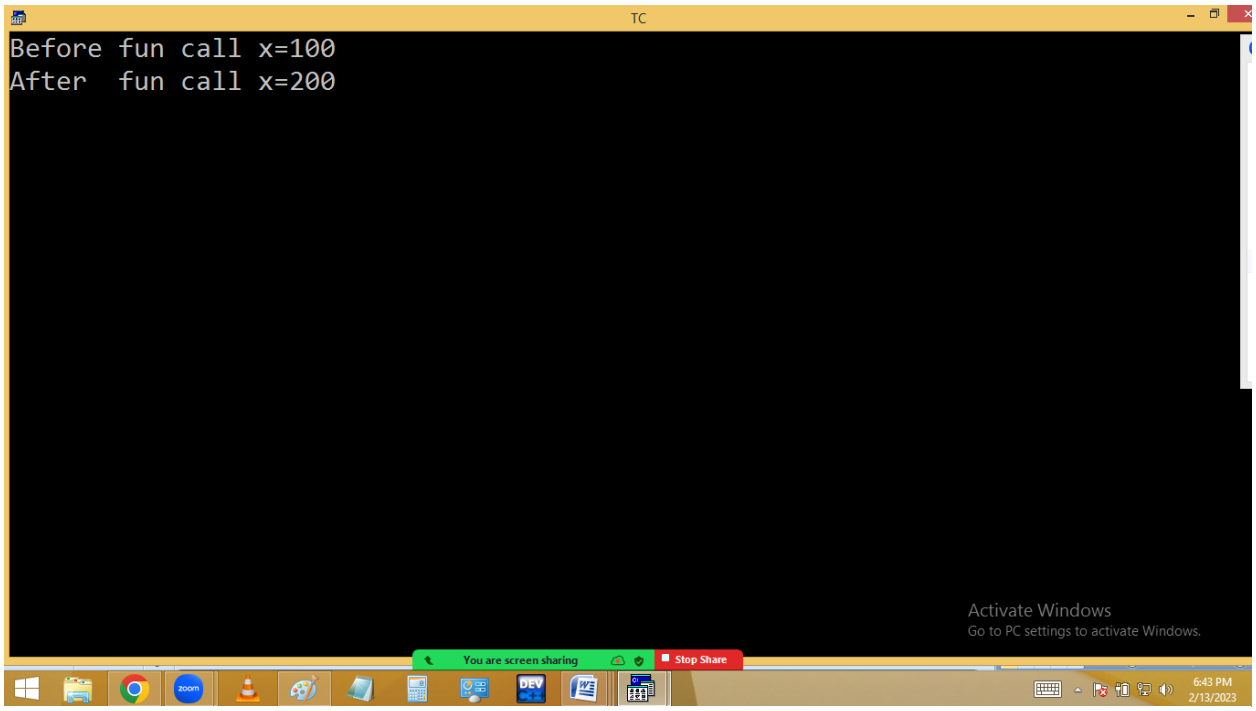
}

Output:

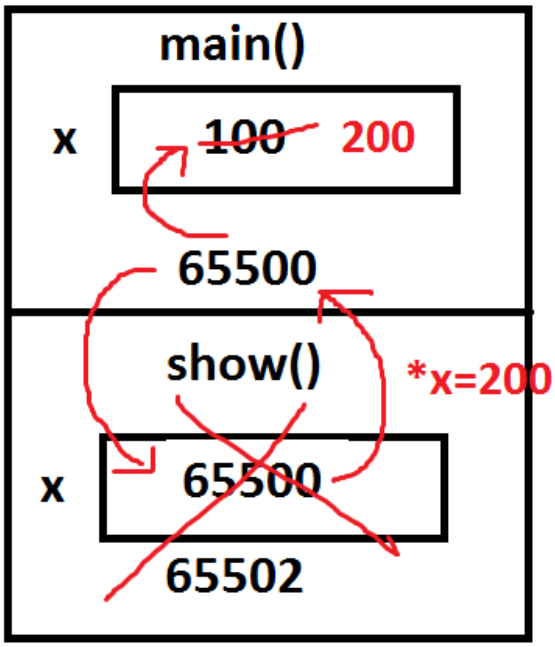
Before function call a=5, b=7

After function call a=7, b=5





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

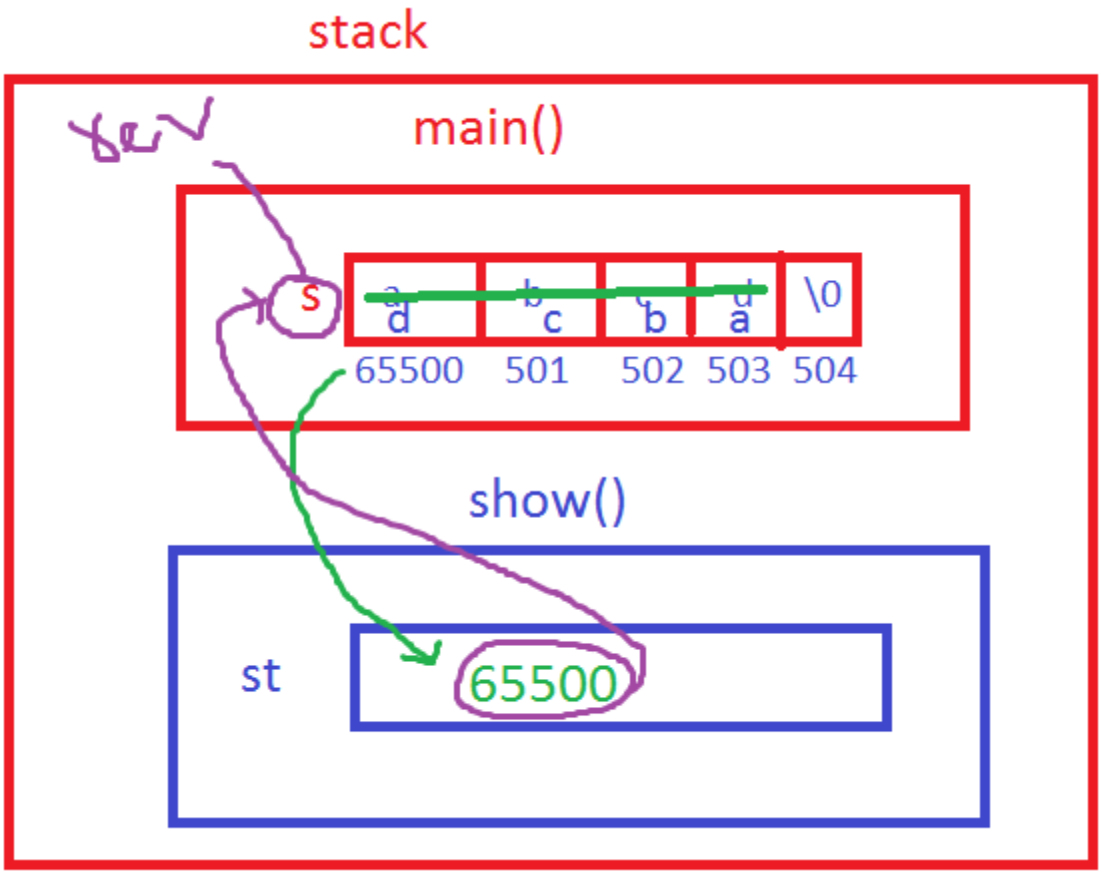
We can pass string / array actual parameter with or without address.

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};
```

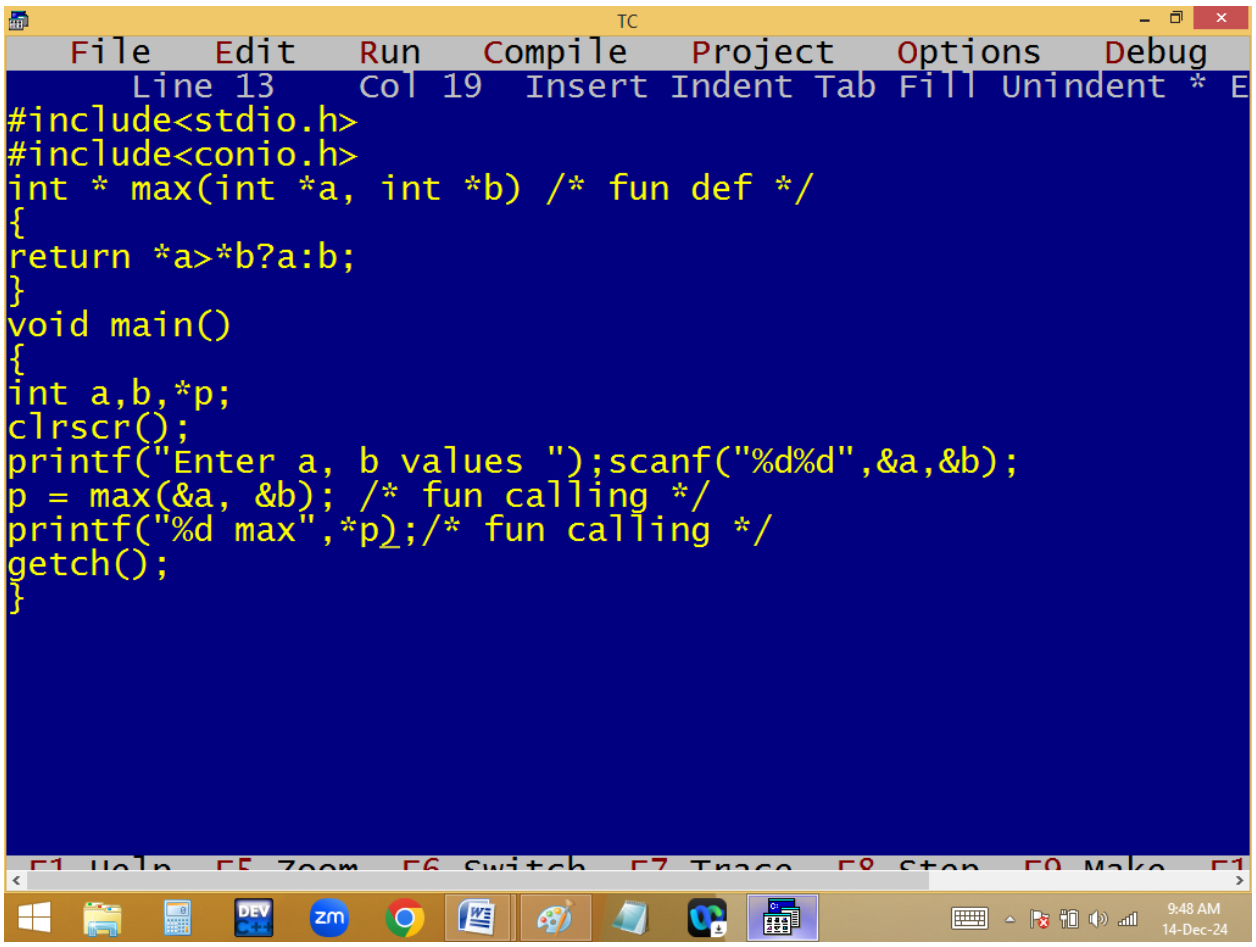
clrscr();

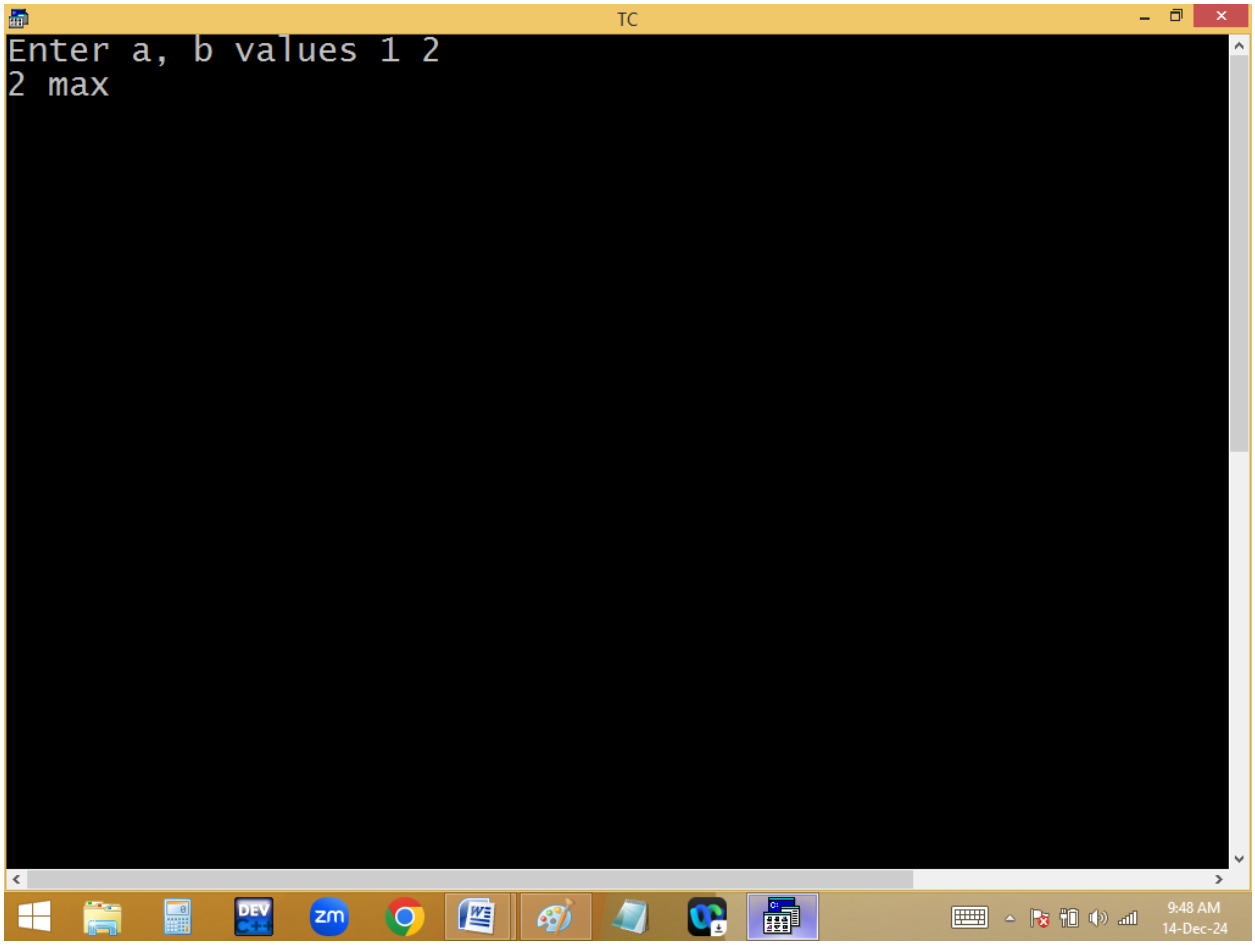
show(a); /* fun calling */

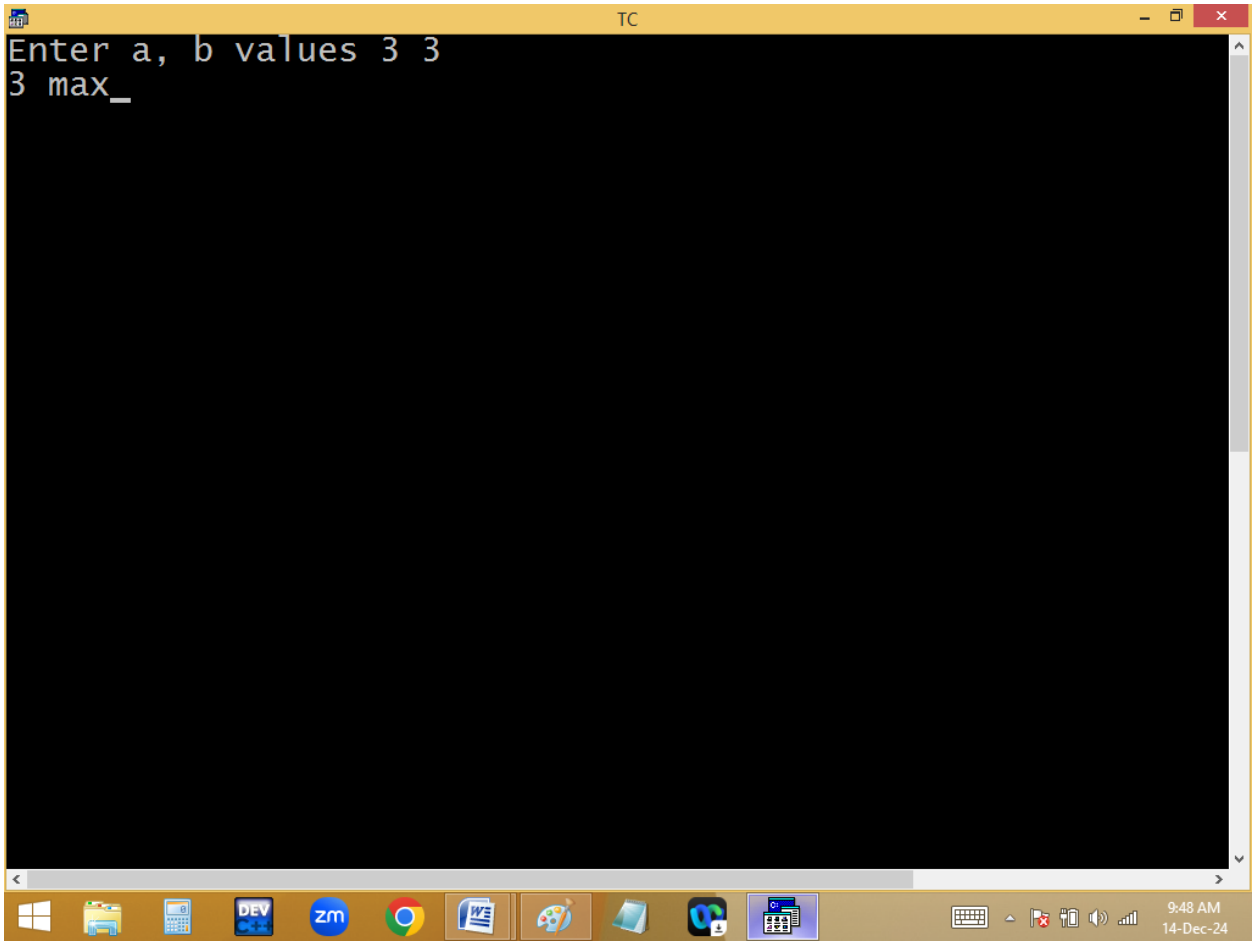
getch();

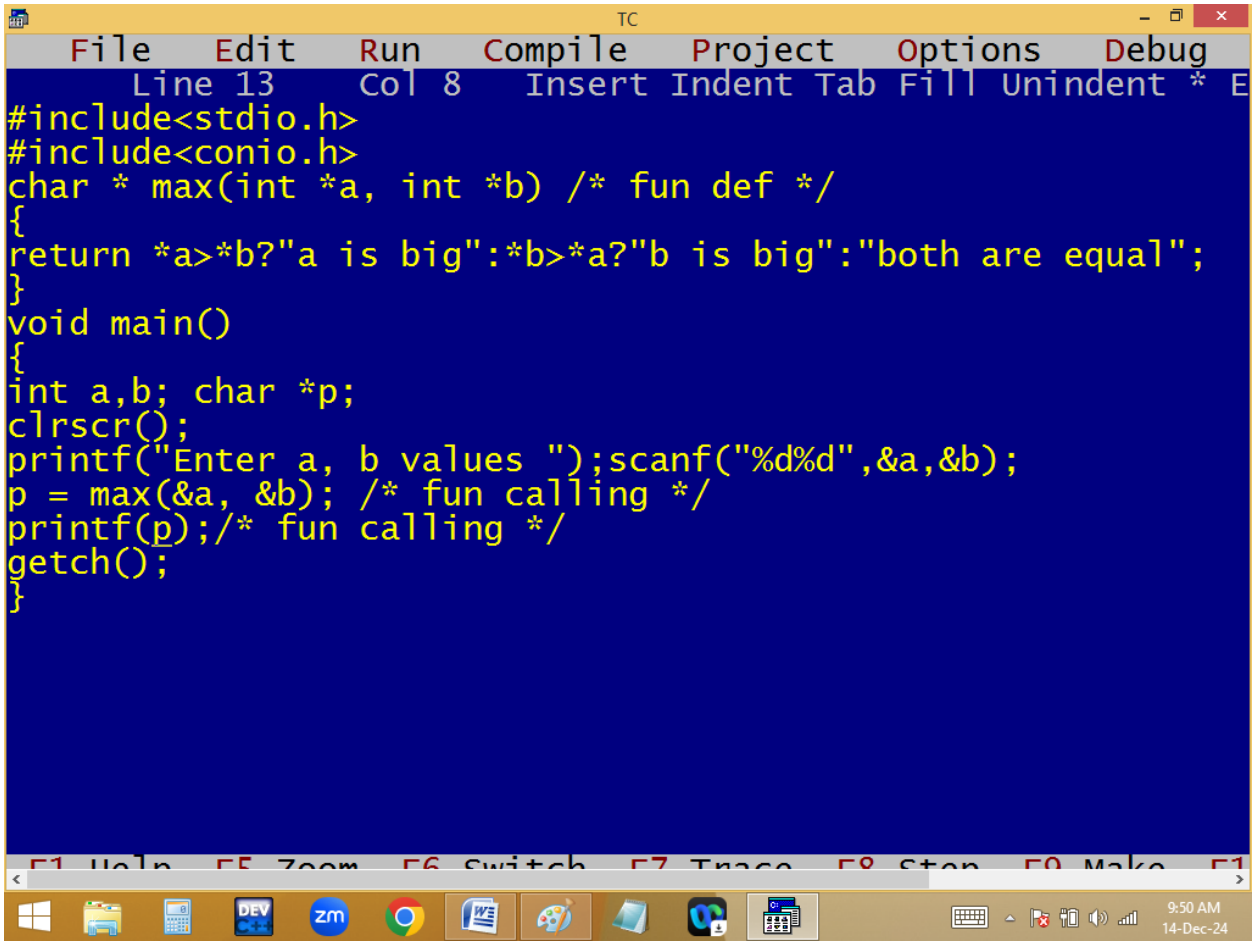
}

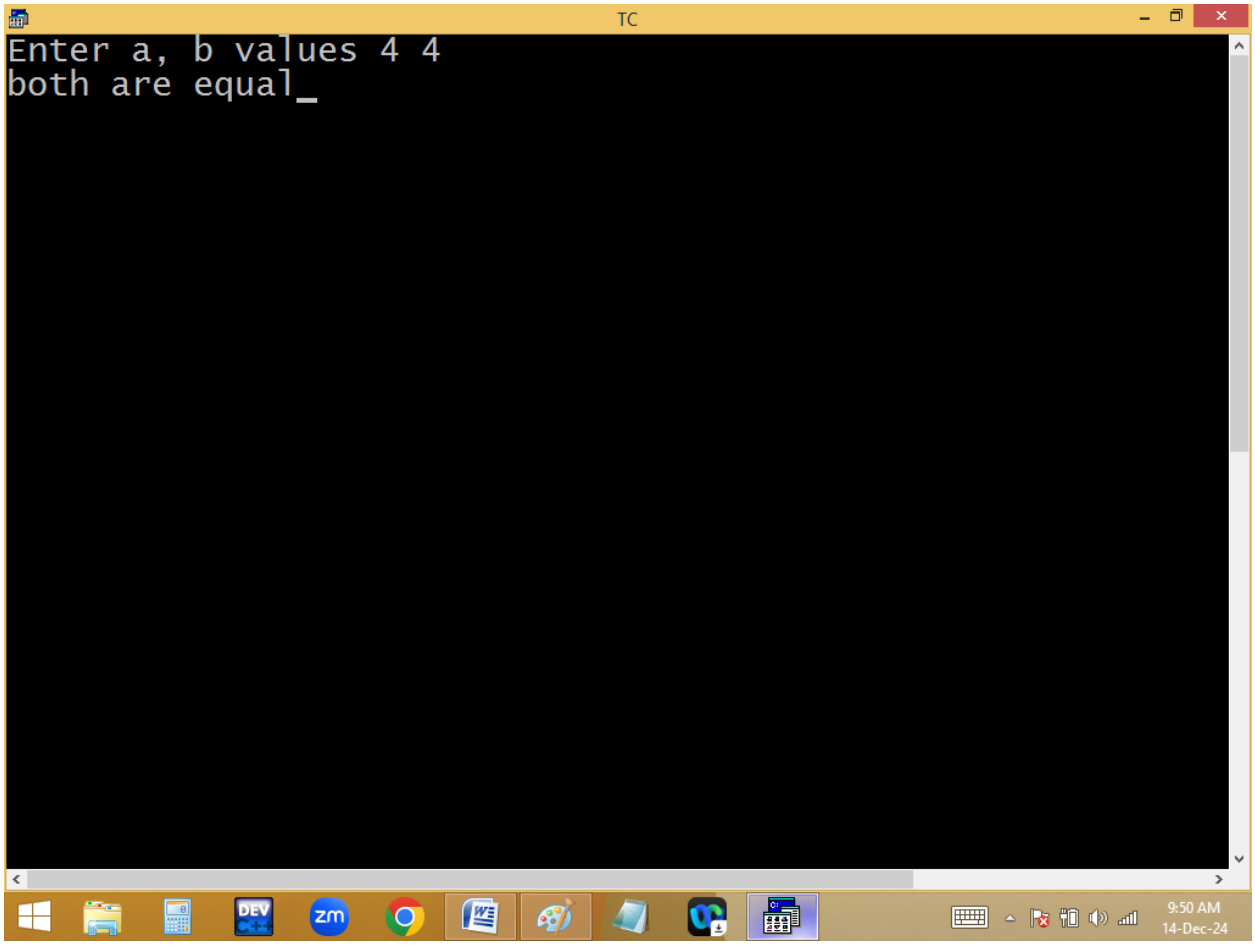
function returning address [pointer]:

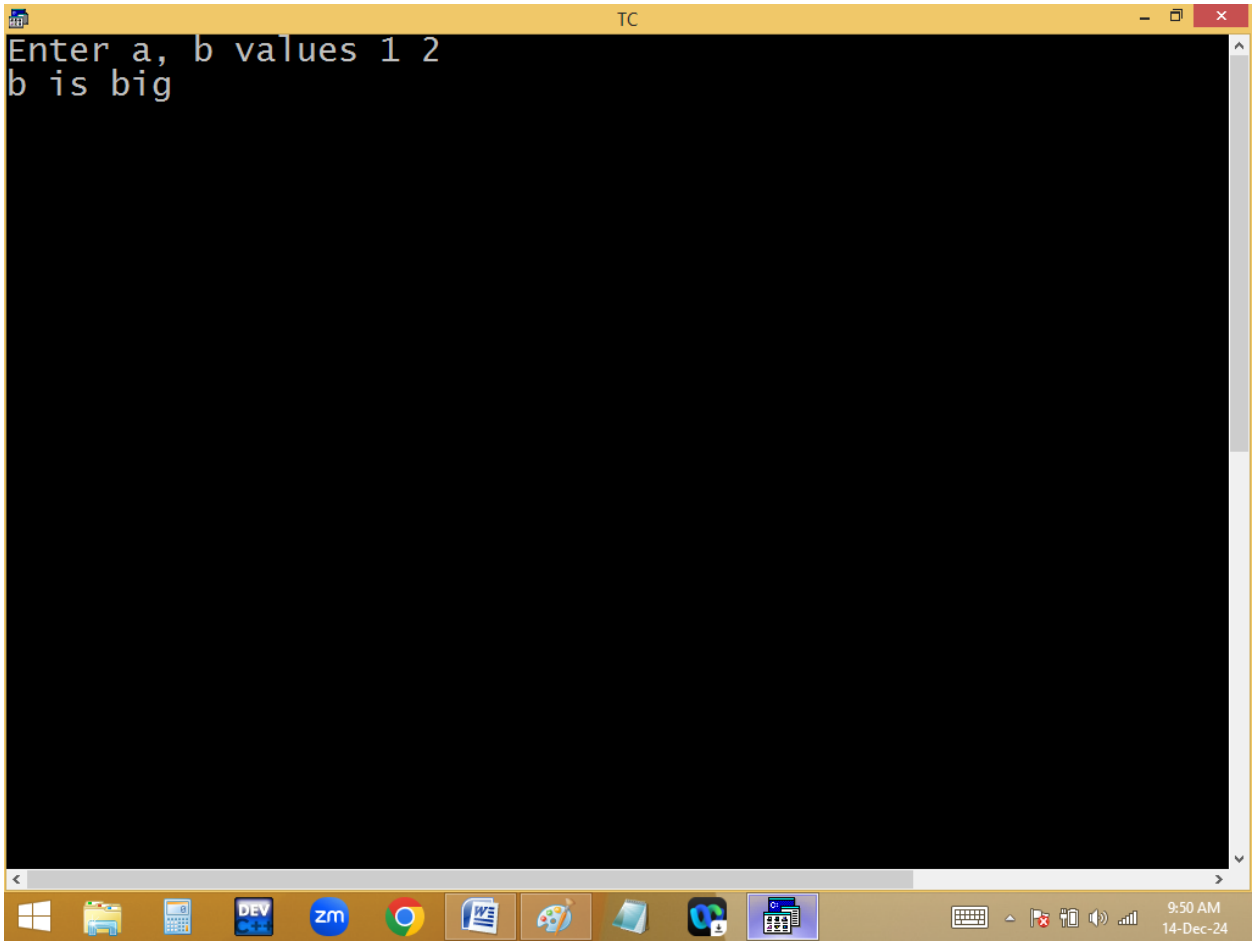


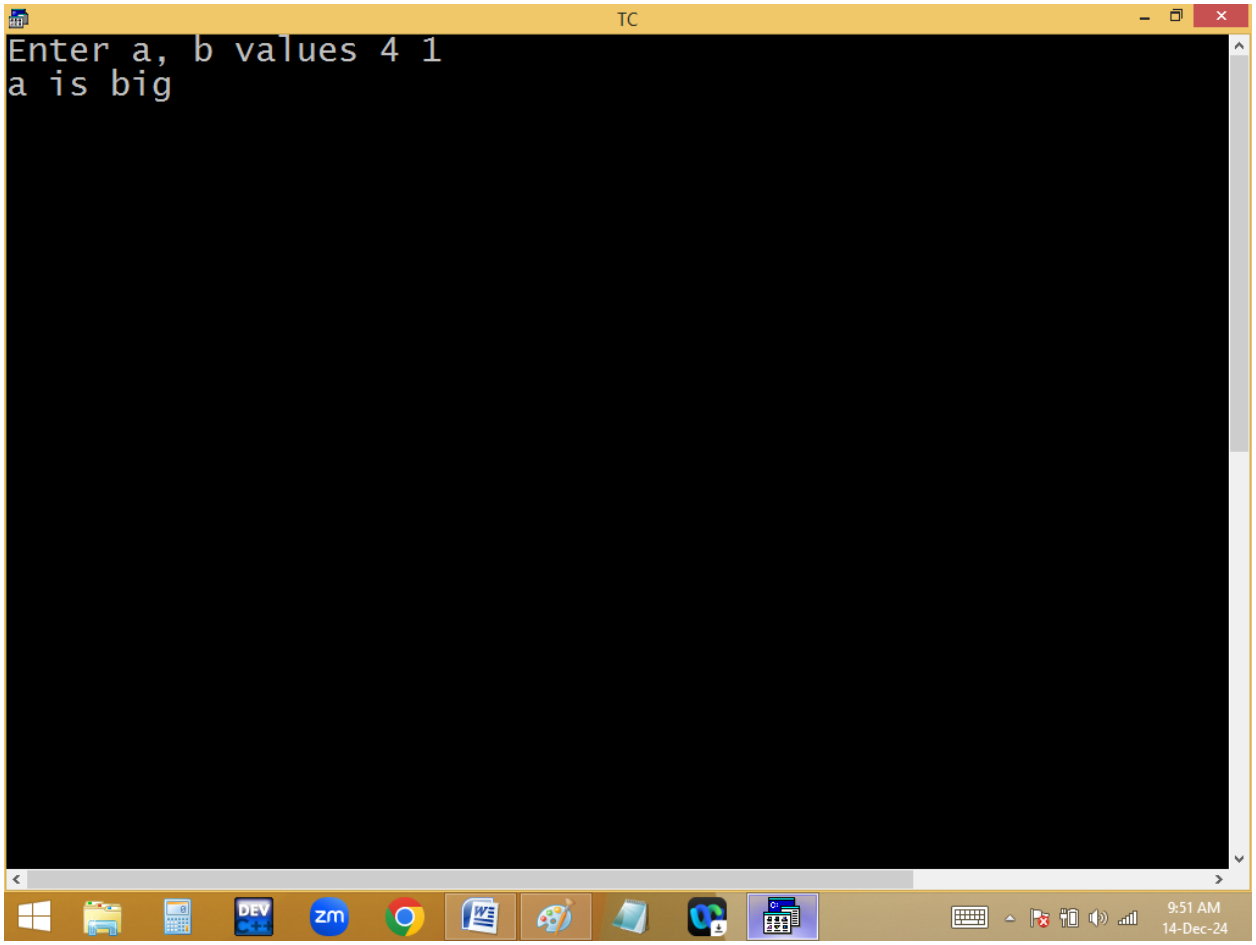

TC
Enter a, b values 1 2
2 max
9:48 AM
14-Dec-24


TC
Enter a, b values 3 3
3 max_
9:48 AM
14-Dec-24


TC
File Edit Run Compile Project Options Debug
Line 13 Col 8 Insert Indent Tab Fill Unindent * E
#include<stdio.h>
#include<conio.h>
char * max(int *a, int *b) /* fun def */
{
return *a>*b?"a is big":*b>*a?"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); /* fun calling */
printf(p);/* fun calling */
getch();
}
9:50 AM
14-Dec-24


TC
Enter a, b values 4 4
both are equal
9:50 AM
14-Dec-24


TC
Enter a, b values 1 2
b is big
9:50 AM
14-Dec-24



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

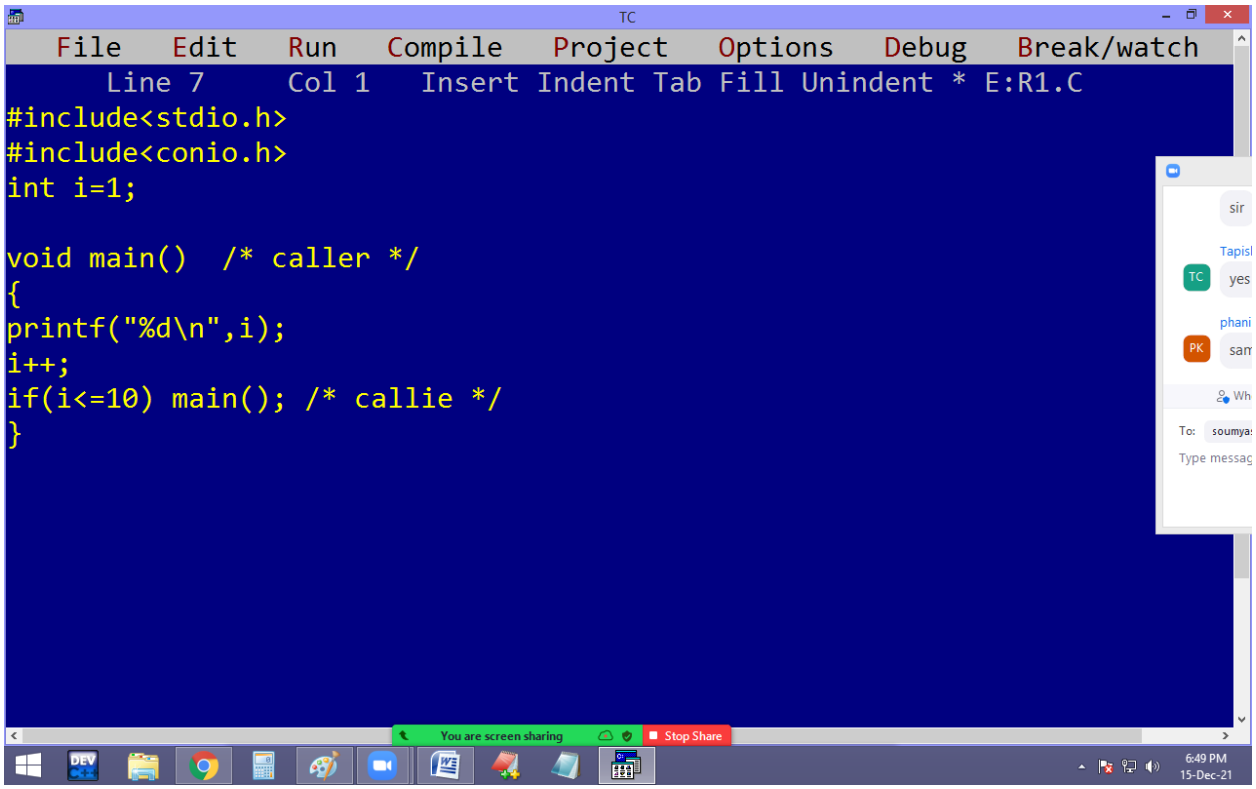
```
#include<stdio.h>
#include<conio.h>

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

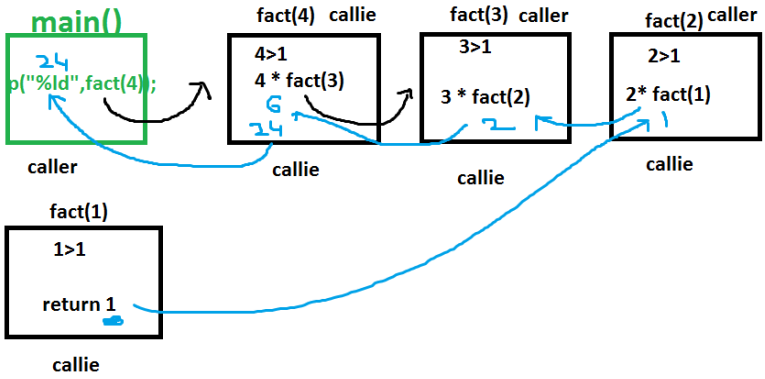
Eg: Finding factorial using recursion:

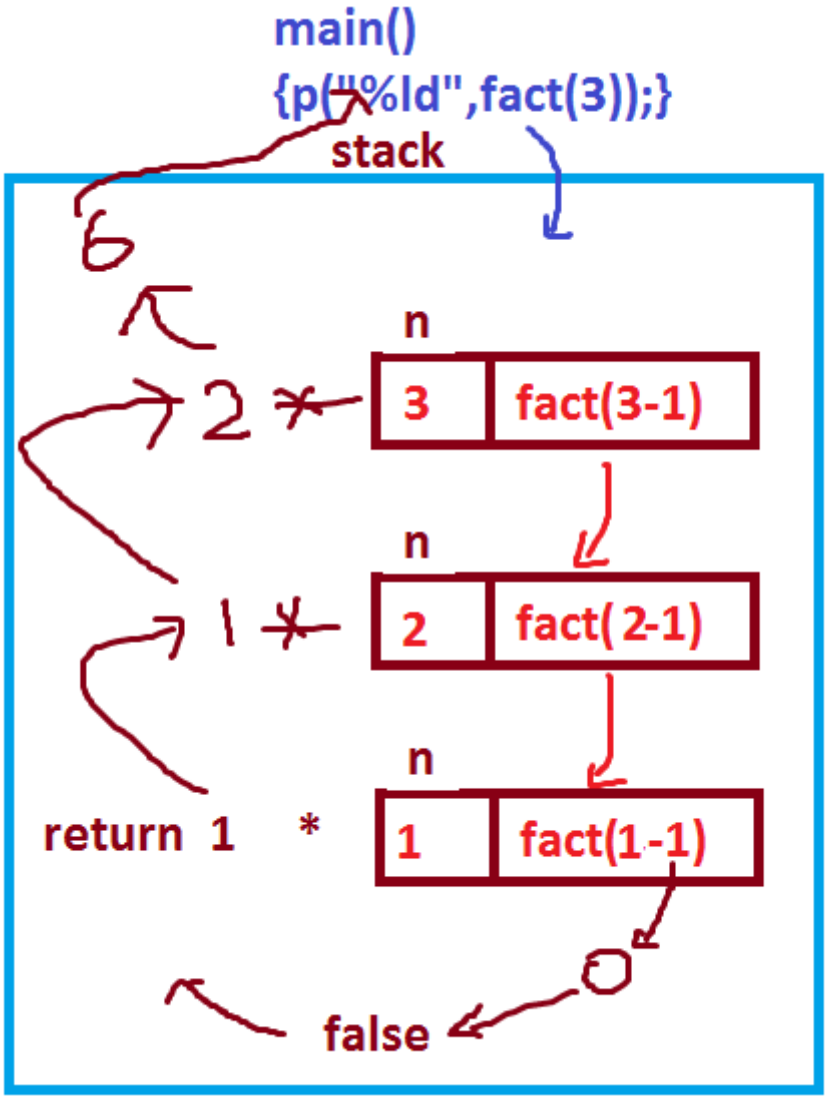
```
#include<stdio.h>
#include<conio.h>
long  fact(int  n)
{
if(n!=0) return  n * fact(n-1);  else  return 1;
}
void  main()
{
int  n;
clrscr();
printf("Enter a no "); scanf("%d", &n);
printf("%d  Factorial = %ld", n, fact(n));
getch();
}
```

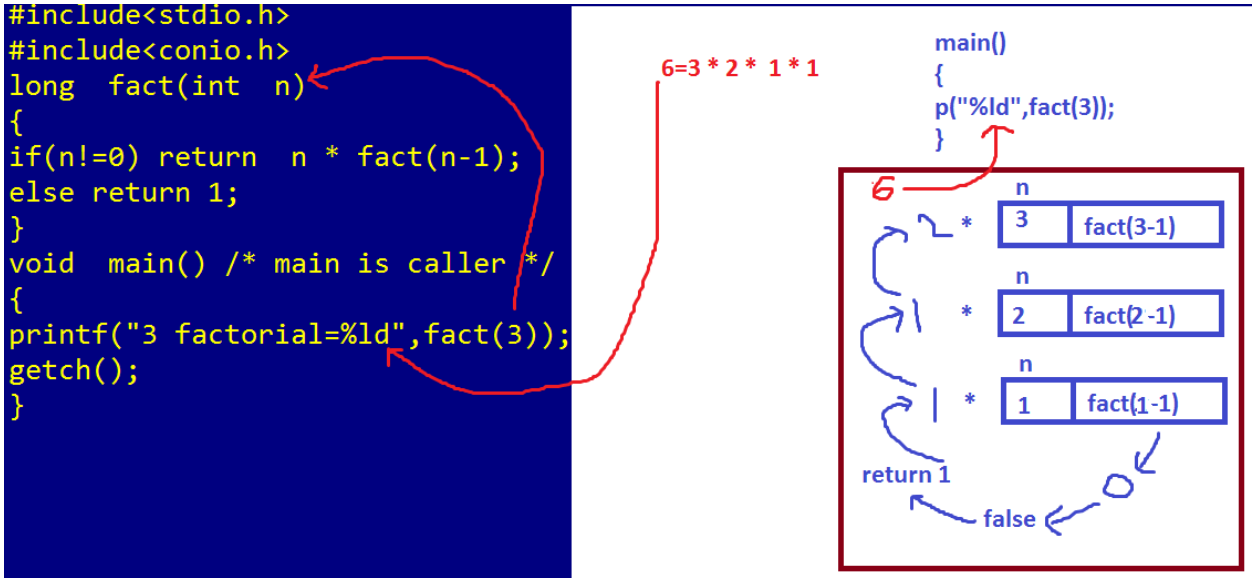
O/P: Enter a no 5

5 Factorial = 120

```
long fact(int n)
{
if(n>1) return n * fact(n-1);
return 1;
}
void main()
{
p("4 factorial  %ld", fact(4));
getch();
}
```







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);

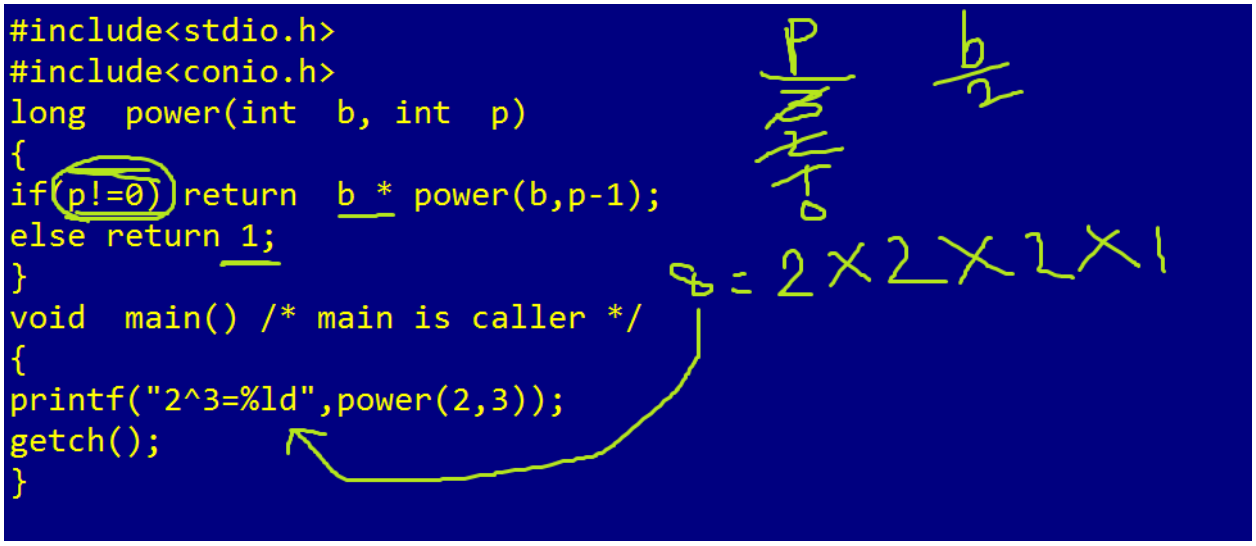
printf(“%d ^ %d = %ld”, b, p, power(b,p));

getch();

}

Output: Enter base, power values 2 5

2 ^ 5 = 32



Eg : Finding digital sum using recursion

#include<stdio.h>

#include<conio.h>

int s=0; /* global var*/

int dsum(long n)

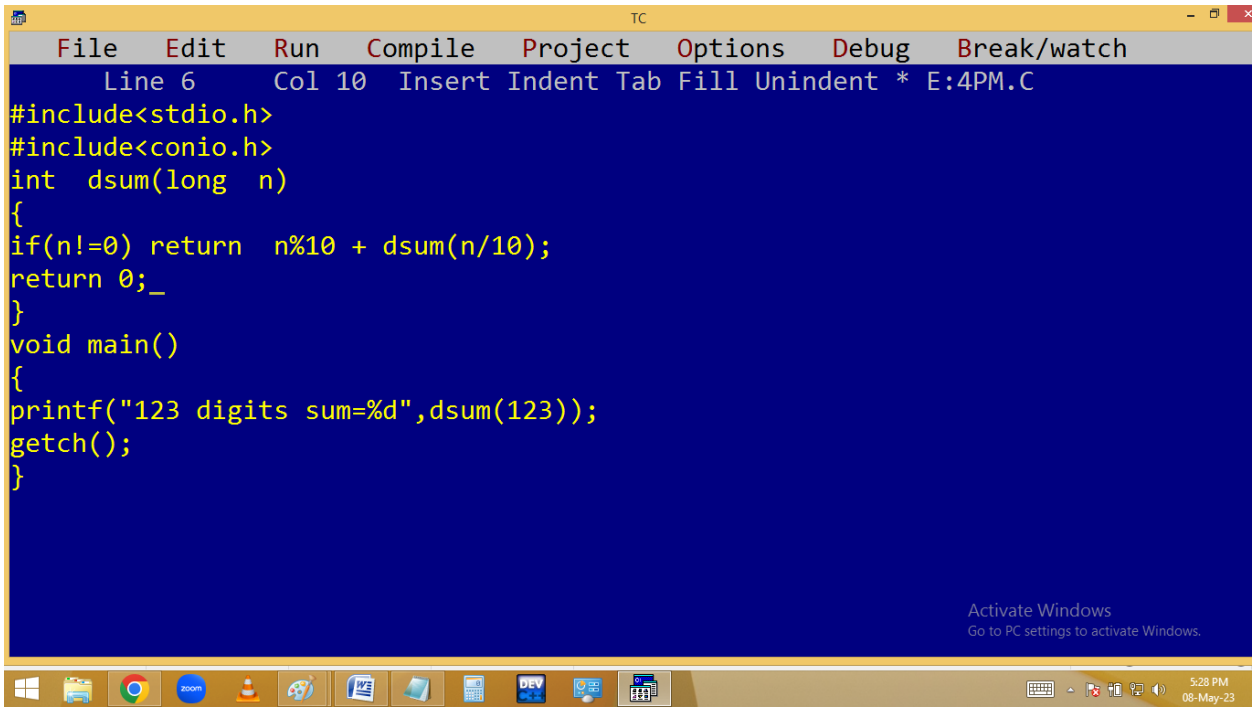
```
{
if(n!=0)
{
s=s+n%10;
dsum(n/10);
}
return s;
}
void main()
{
long  n;
clrscr();
printf("Enter a no");
scanf("%ld",&n);
printf("%ld digital sum = %d",n,dsum(n));
getch();
}
```

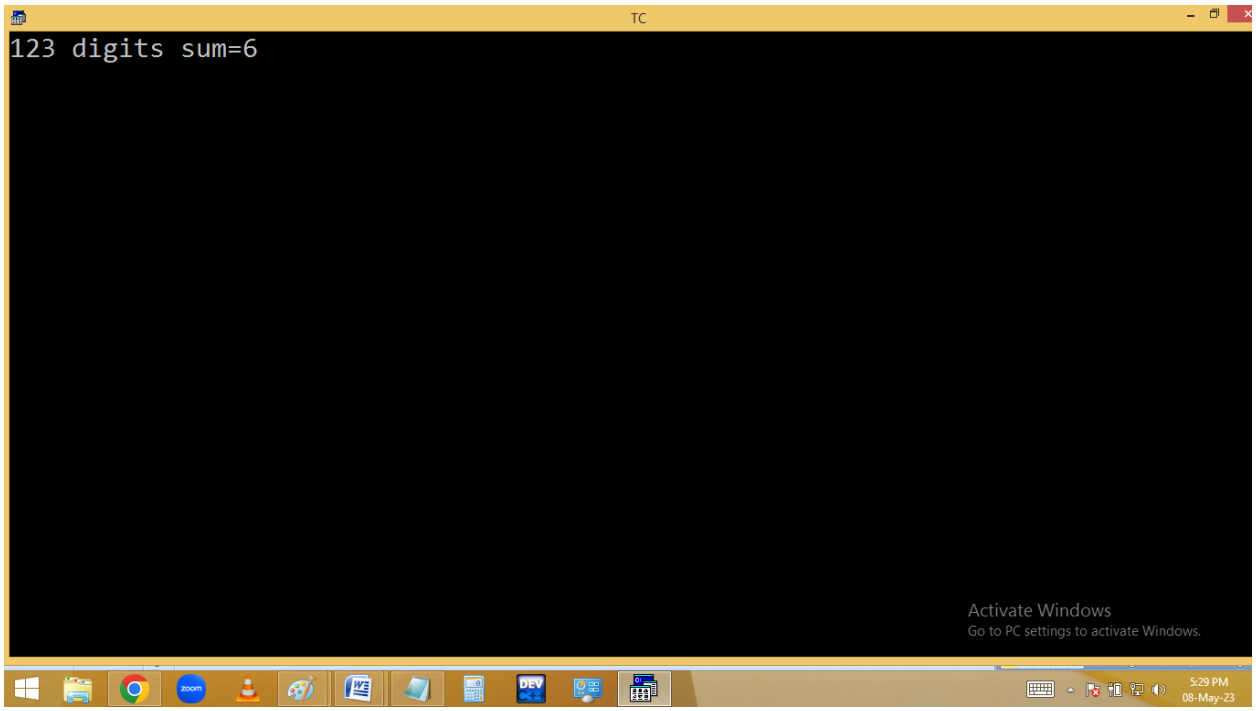
Output:

Enter a no: 123

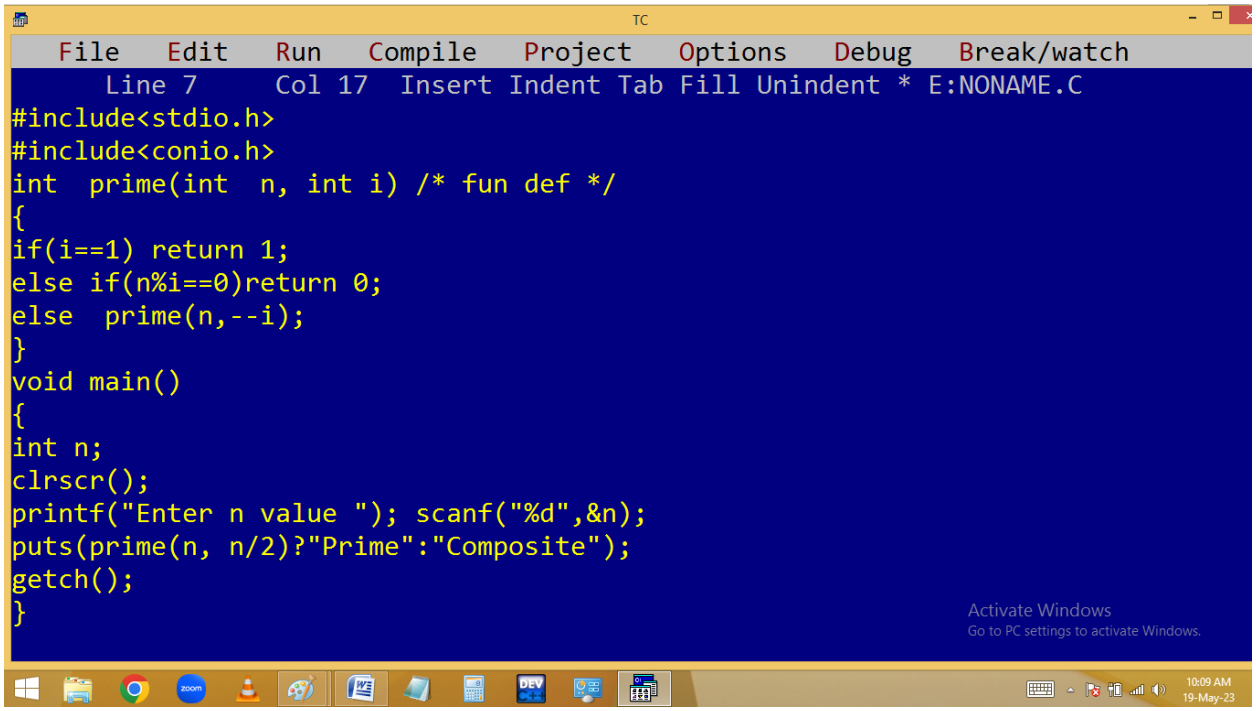
123 digital sum = 6

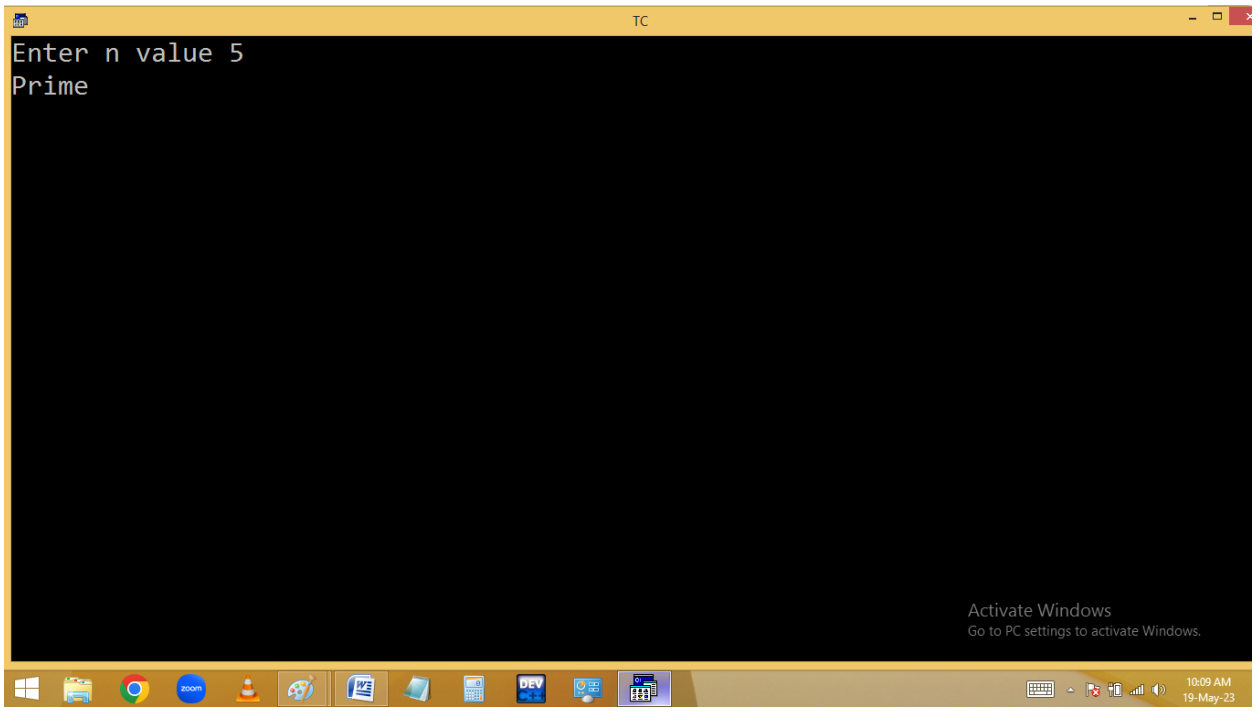
```
#include<stdio.h>
#include<conio.h>
int dsum(long int  n)
{
static int s;
if(n!=0)
{
s+=n%10;  dsum(n/10);
}
return  s;
}
void  main() /* main is caller */
{
printf("123 digital sum=%d",dsum(123));
getch();
}
```

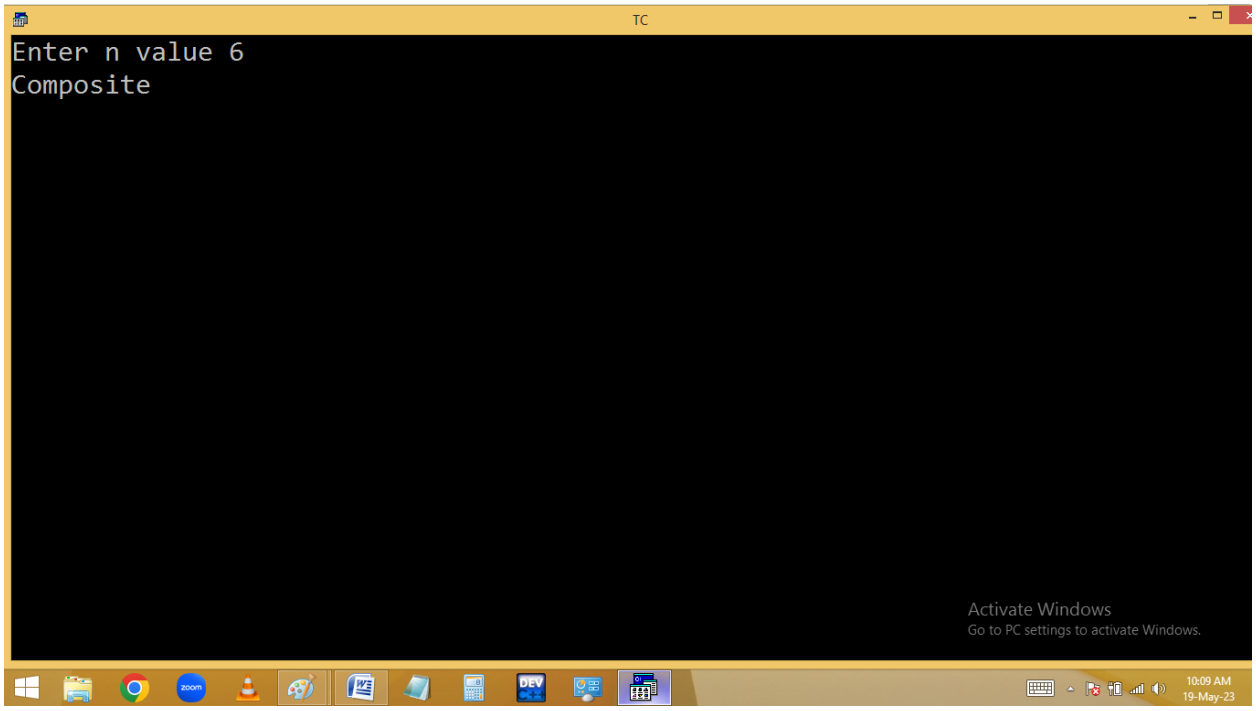



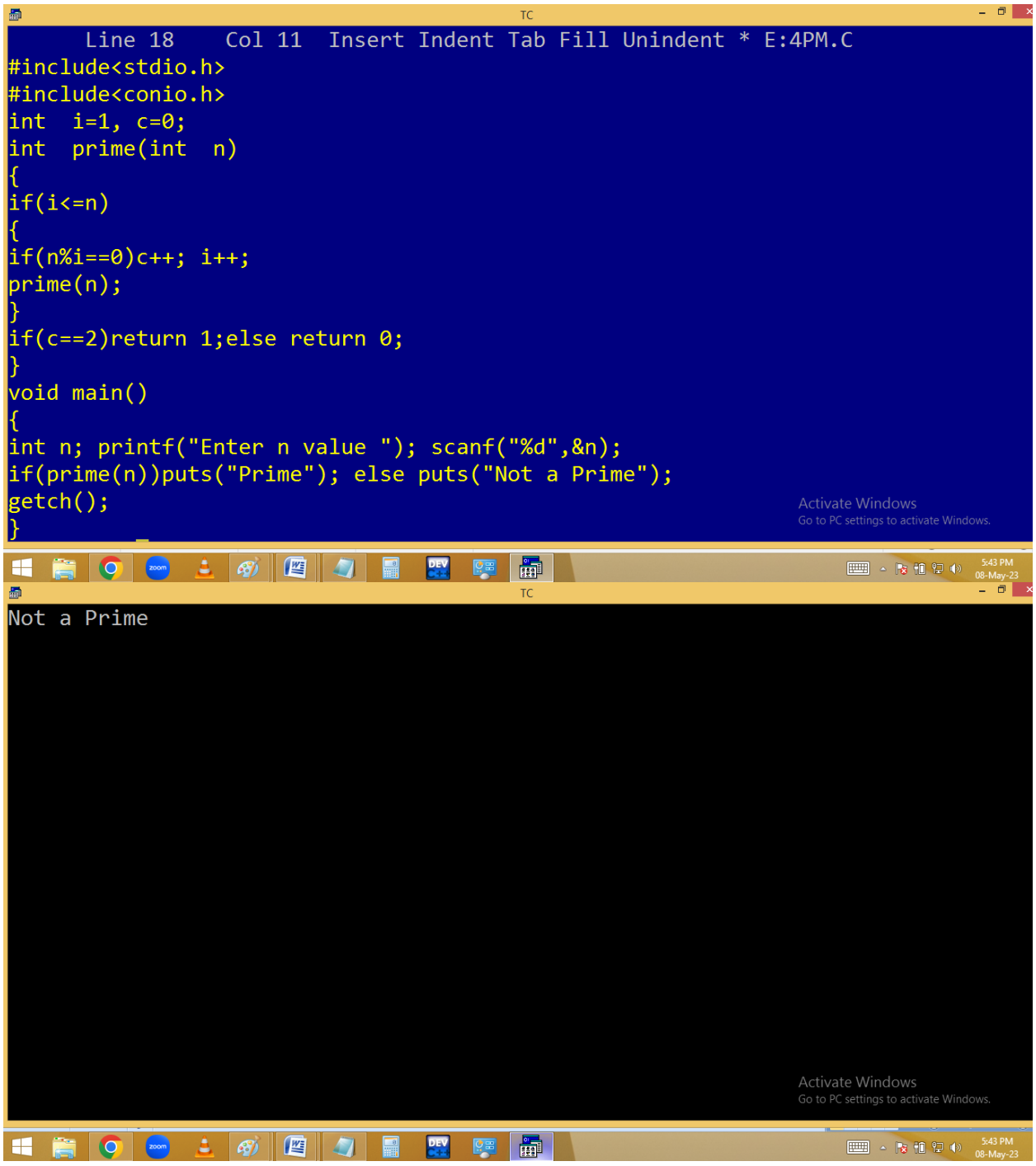


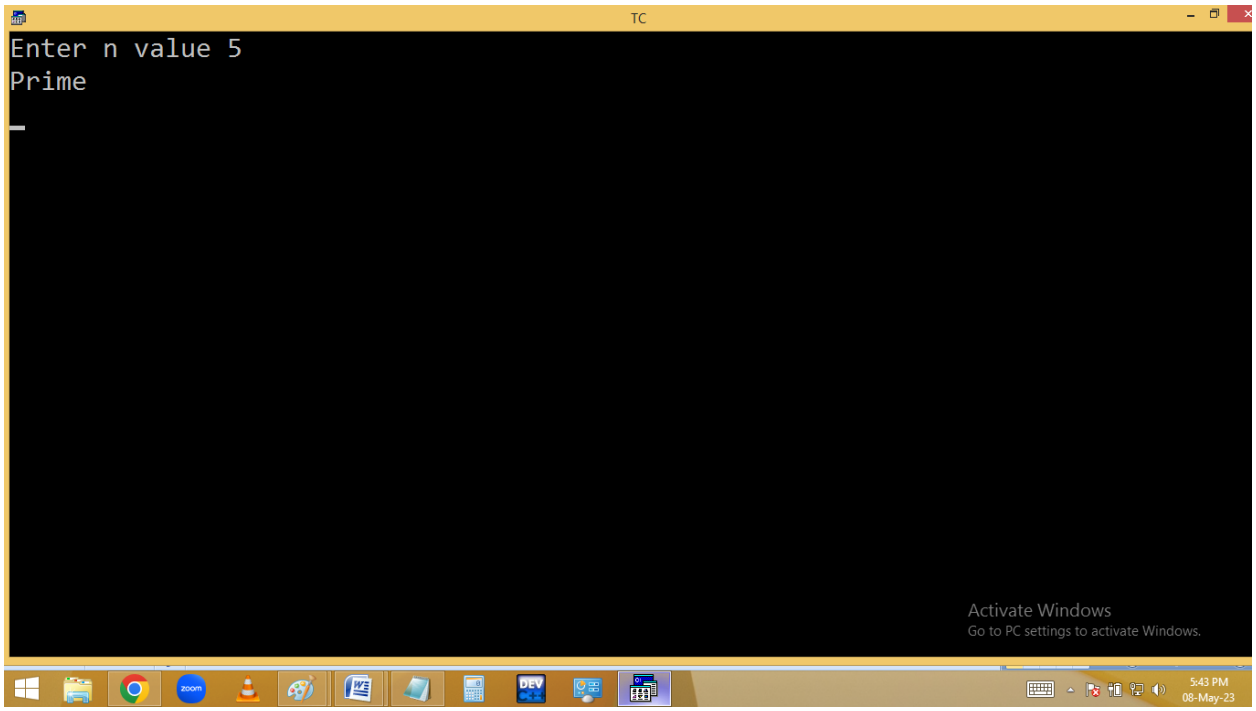
Finding prime using recursion:




TC
Enter n value 5
Prime
Activate Windows
Go to PC settings to activate Windows.
10:09 AM
19-May-23


TC
Enter n value 6
Composite
Activate Windows
Go to PC settings to activate Windows.
10:09 AM
19-May-23


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();
}
Activate Windows
Go to PC settings to activate Windows.
5:43 PM
08-May-23
TC
Not a Prime
Activate Windows
Go to PC settings to activate Windows.
5:43 PM
08-May-23


TC
Enter n value 5
Prime
Activate Windows
Go to PC settings to activate Windows.
5:43 PM
08-May-23