

## **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 if 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.

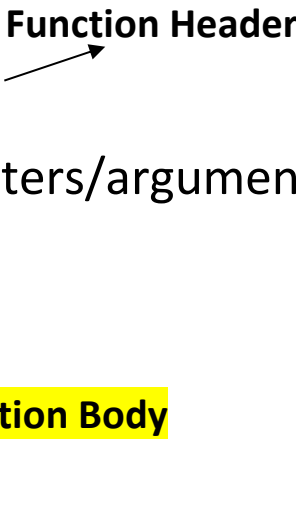
Function body consists of statements related to the task of function.

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:**

[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.

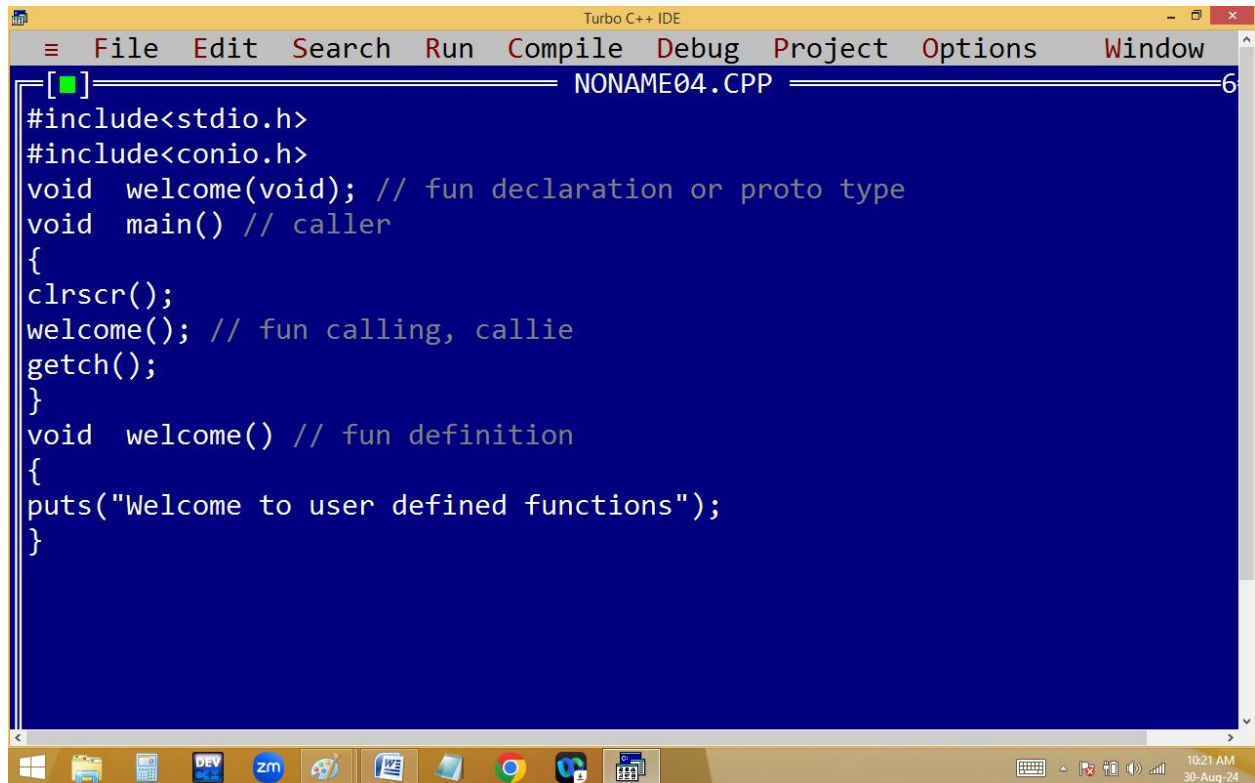
Difference between arguments and parameters is “arguments are nothing but different vehicles which are parked in same parking area and the parking area is the parameters. i.e. arguments are always changed but parameters are fixed.

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

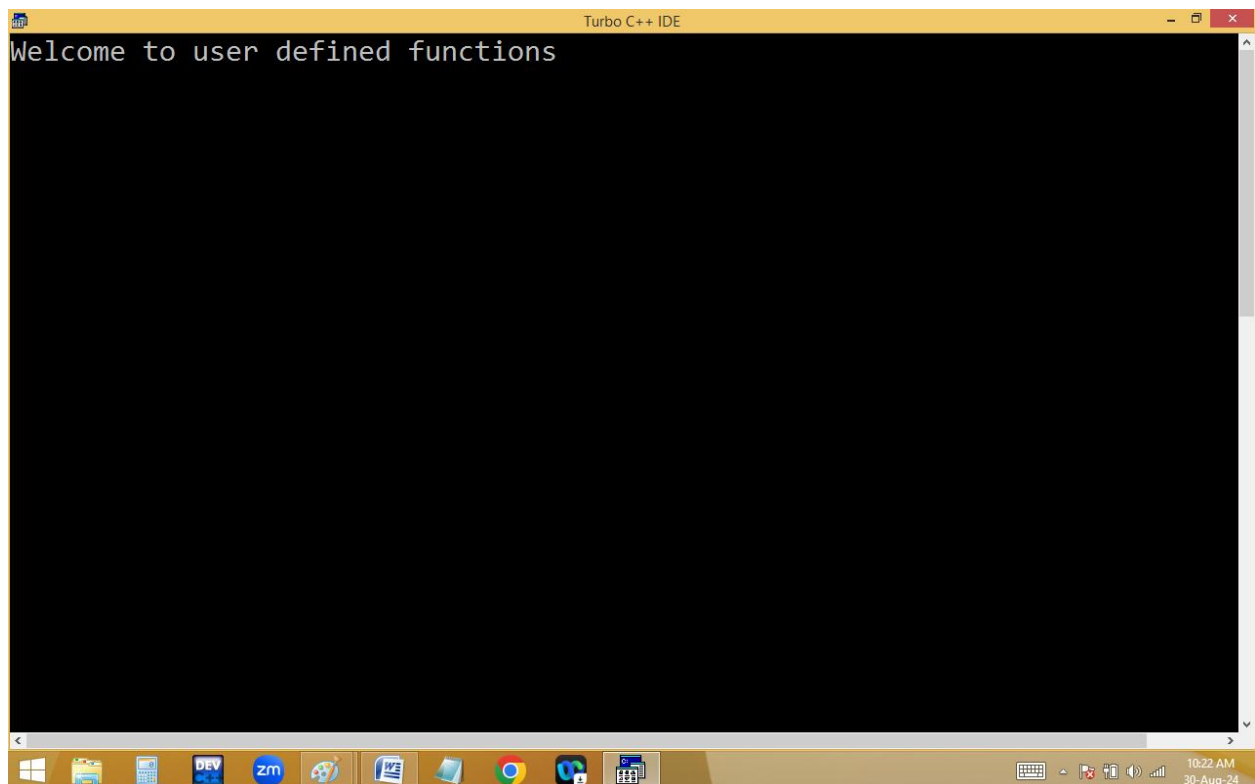
Based on arguments and return values, functions are divided into 4 types.

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.

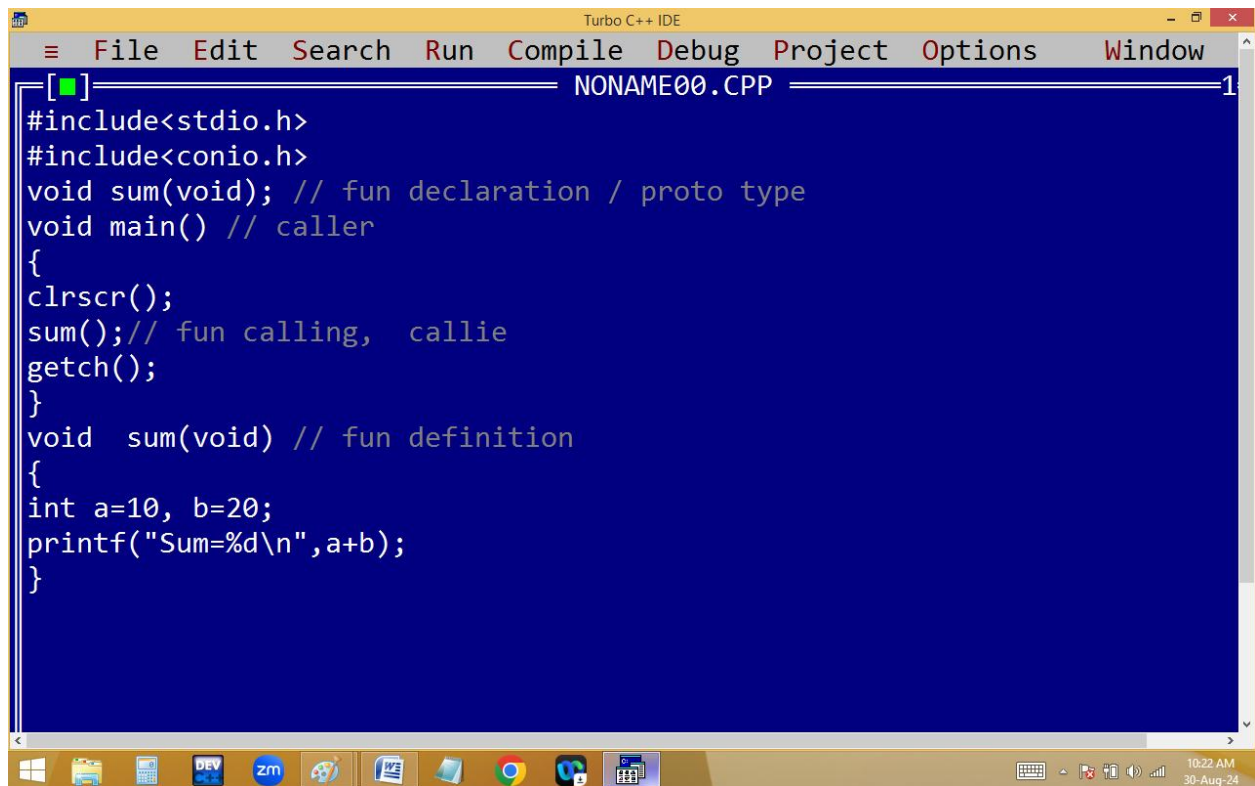
## Function without arguments, without return values:



```
Turbo C++ IDE
File Edit Search Run Compile Debug Project Options Window
[ ] NONAME04.CPP 6
#include<stdio.h>
#include<conio.h>
void welcome(void); // fun declaration or proto type
void main() // caller
{
clrscr();
welcome(); // fun calling, callie
getch();
}
void welcome() // fun definition
{
puts("Welcome to user defined functions");
}
```

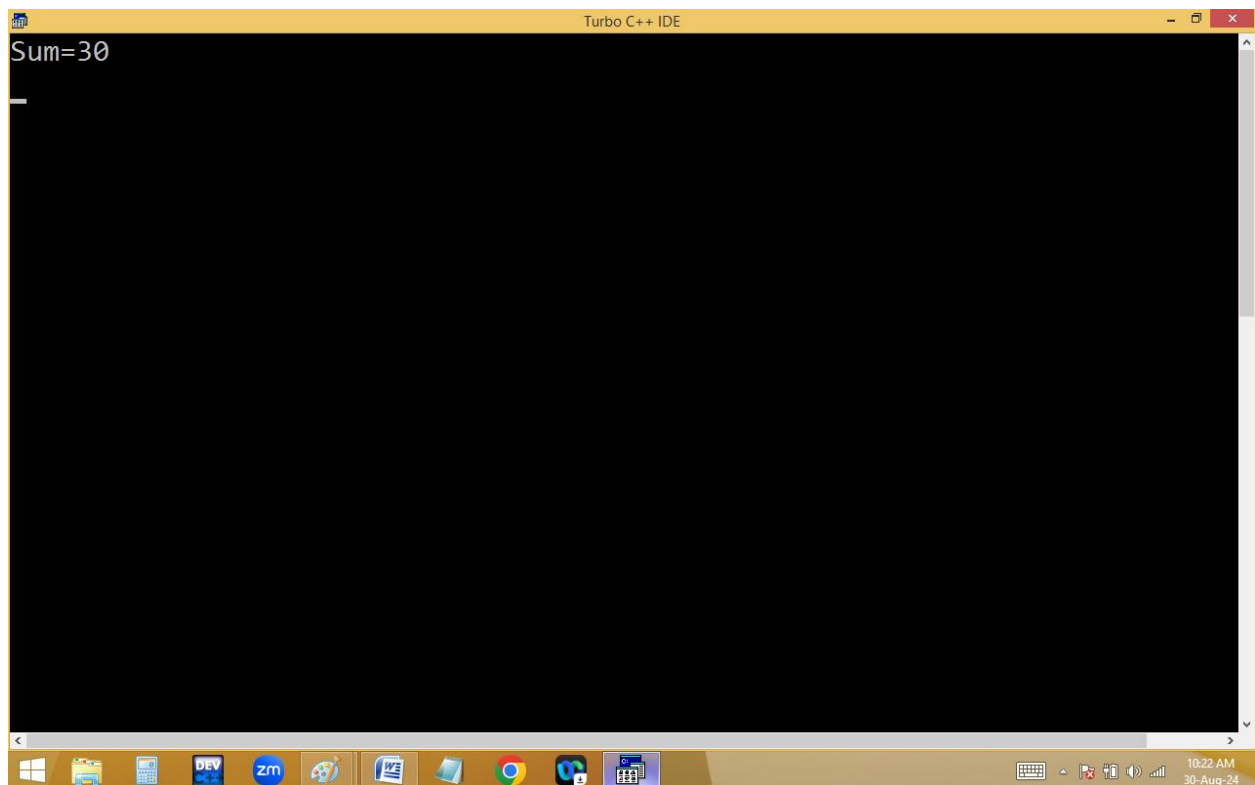


```
Turbo C++ IDE
Welcome to user defined functions
```



The screenshot shows the Turbo C++ IDE with a menu bar (File, Edit, Search, Run, Compile, Debug, Project, Options, Window) and a toolbar. The main window displays a C program in a file named NONAME00.CPP. The code includes headers for stdio.h and conio.h, declares a void sum() function, and defines a main() function that calls clrscr(), sum(), and getch(). The sum() function is defined to take no arguments and prints the sum of a=10 and b=20. The Windows taskbar at the bottom shows various icons including Windows, File Explorer, DEV, zm, and others, with a system clock indicating 10:22 AM on 30-Aug-24.

```
[■] NONAME00.CPP 1
#include<stdio.h>
#include<conio.h>
void sum(void); // fun declaration / proto type
void main() // caller
{
    clrscr();
    sum();// fun calling, callie
    getch();
}
void sum(void) // fun definition
{
    int a=10, b=20;
    printf("Sum=%d\n",a+b);
}
```

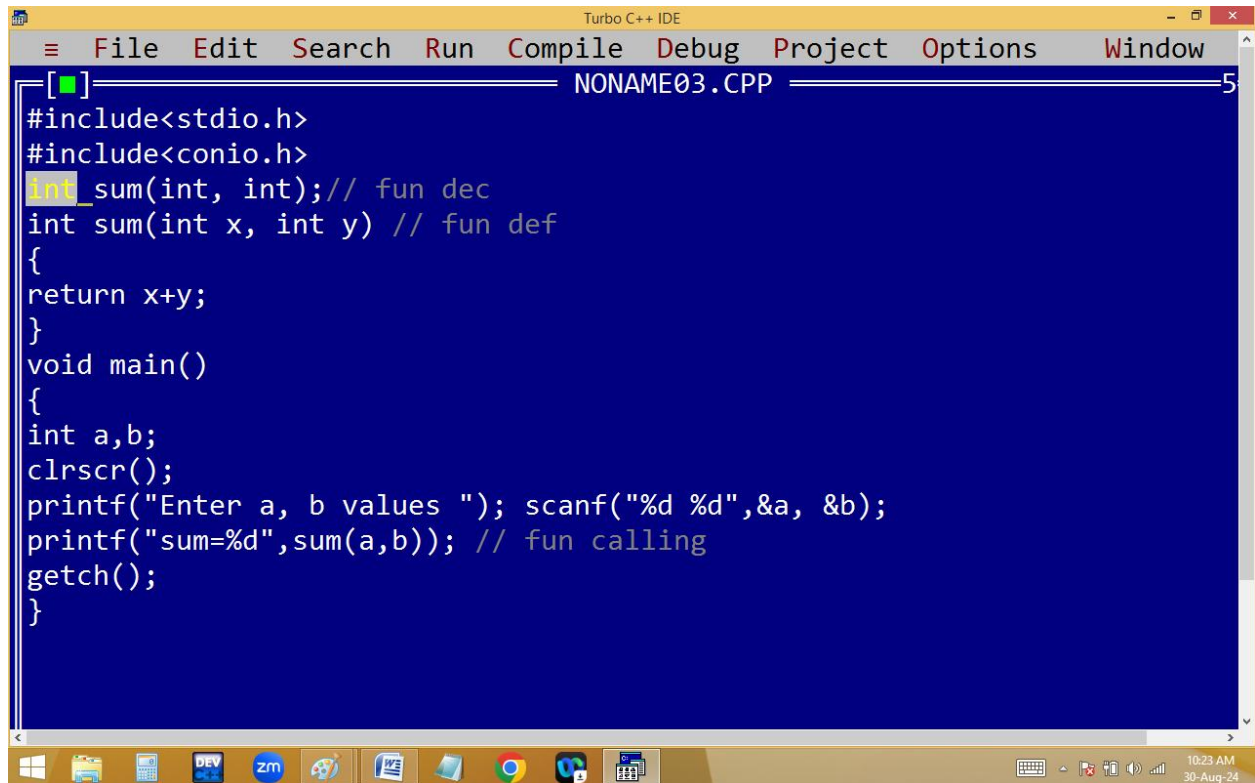


The screenshot shows the Turbo C++ IDE with the same menu bar and toolbar. The main window now displays the output of the program, which is "Sum=30". The Windows taskbar at the bottom is identical to the first screenshot, showing the same icons and system clock.

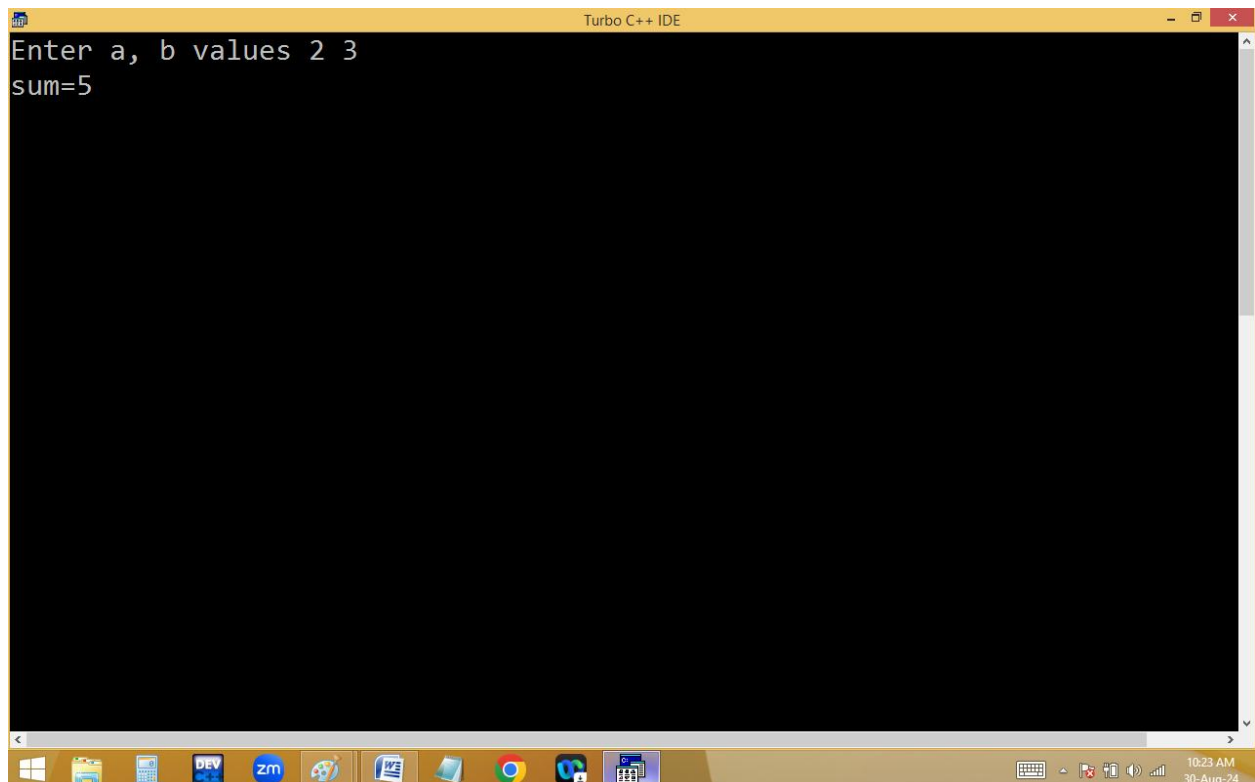
```
Sum=30
```



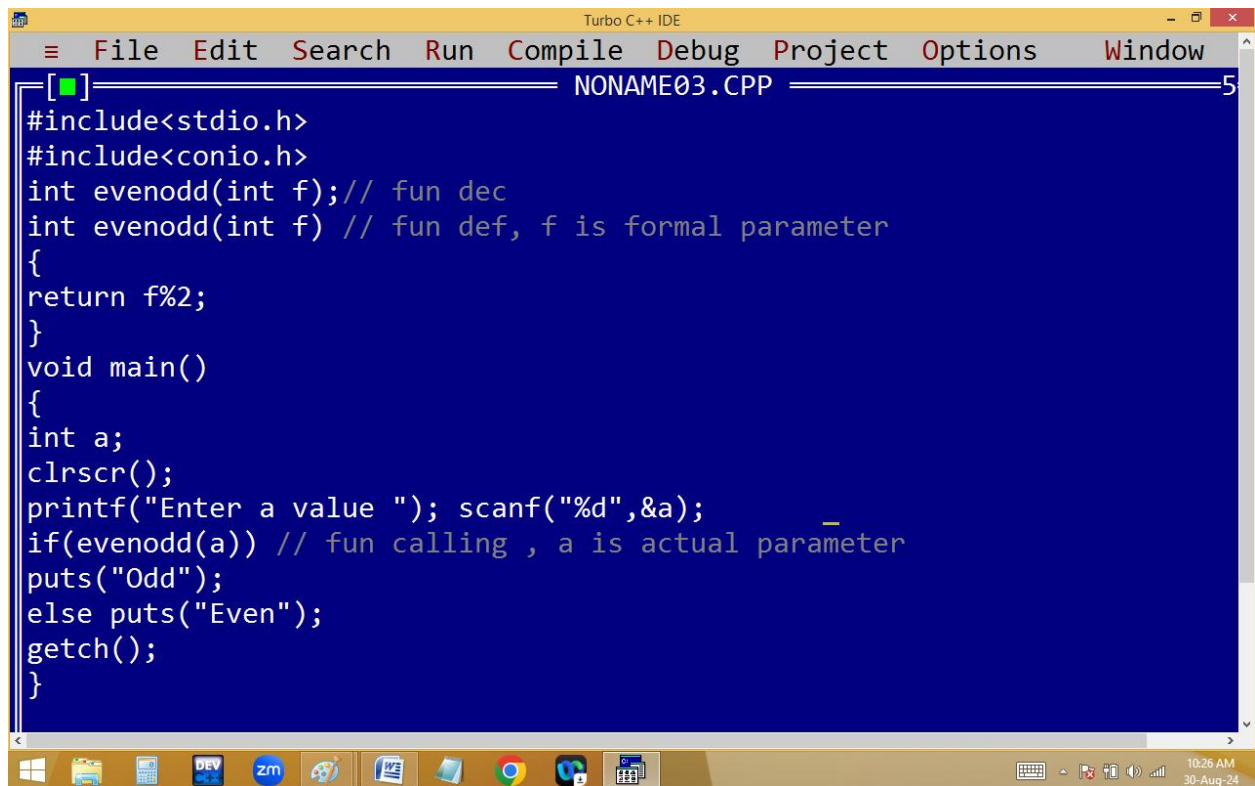
## Function with arguments, with return value:



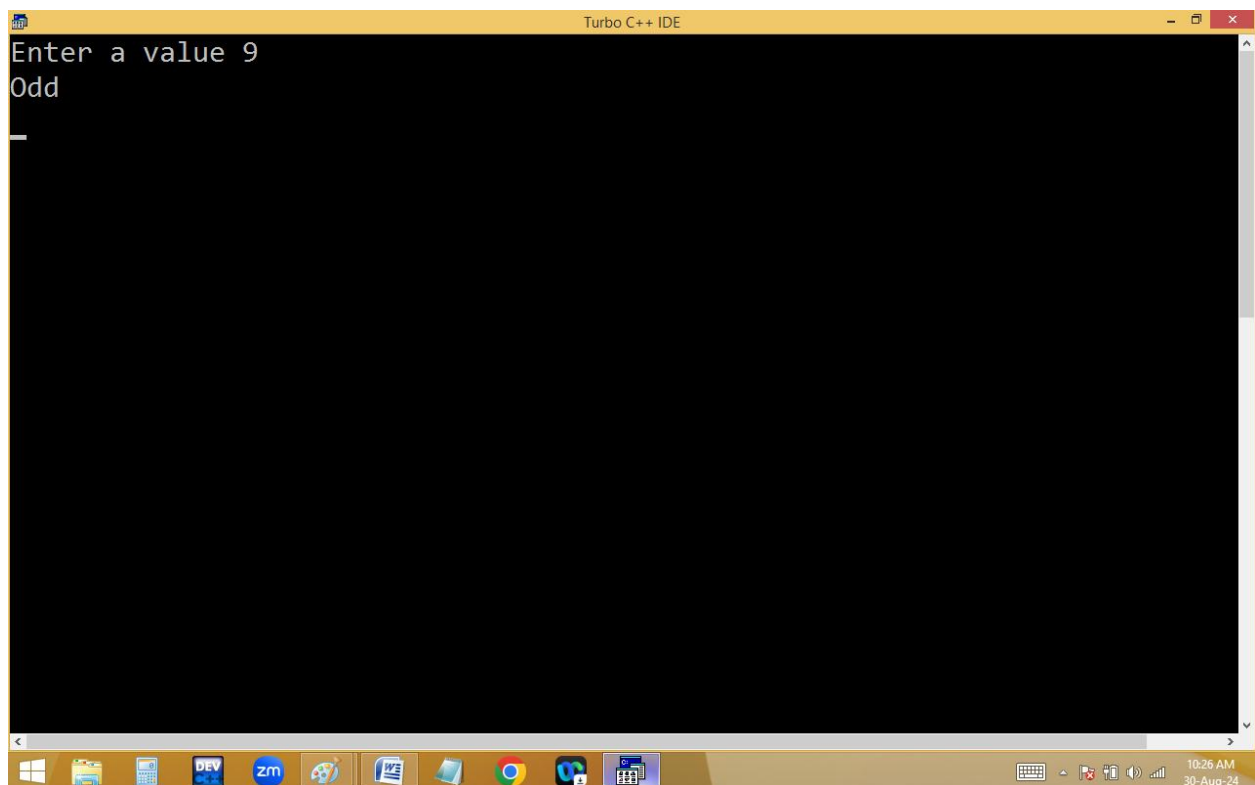
```
Turbo C++ IDE
File Edit Search Run Compile Debug Project Options Window
NONAME03.CPP 5
#include<stdio.h>
#include<conio.h>
int sum(int, int); // fun dec
int sum(int x, int y) // fun def
{
    return x+y;
}
void main()
{
    int a,b;
    clrscr();
    printf("Enter a, b values "); scanf("%d %d",&a, &b);
    printf("sum=%d",sum(a,b)); // fun calling
    getch();
}
```



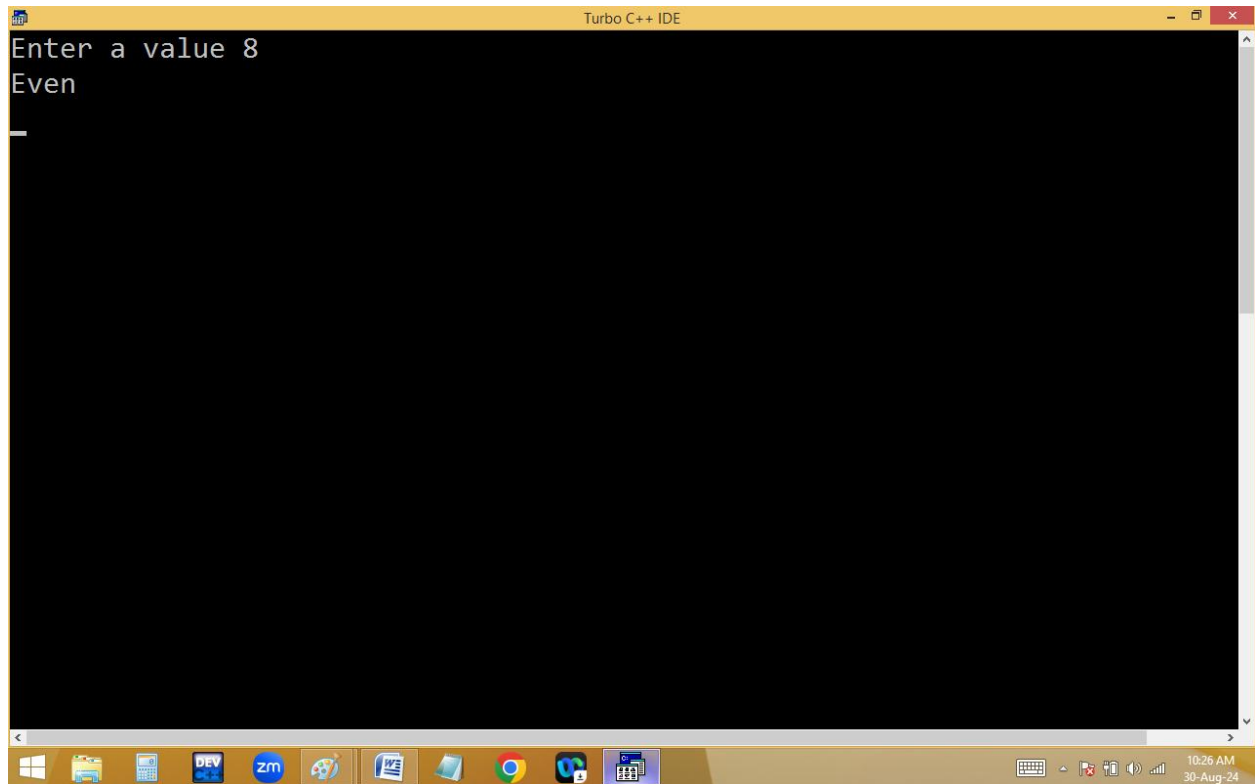
```
Turbo C++ IDE
Enter a, b values 2 3
sum=5
```



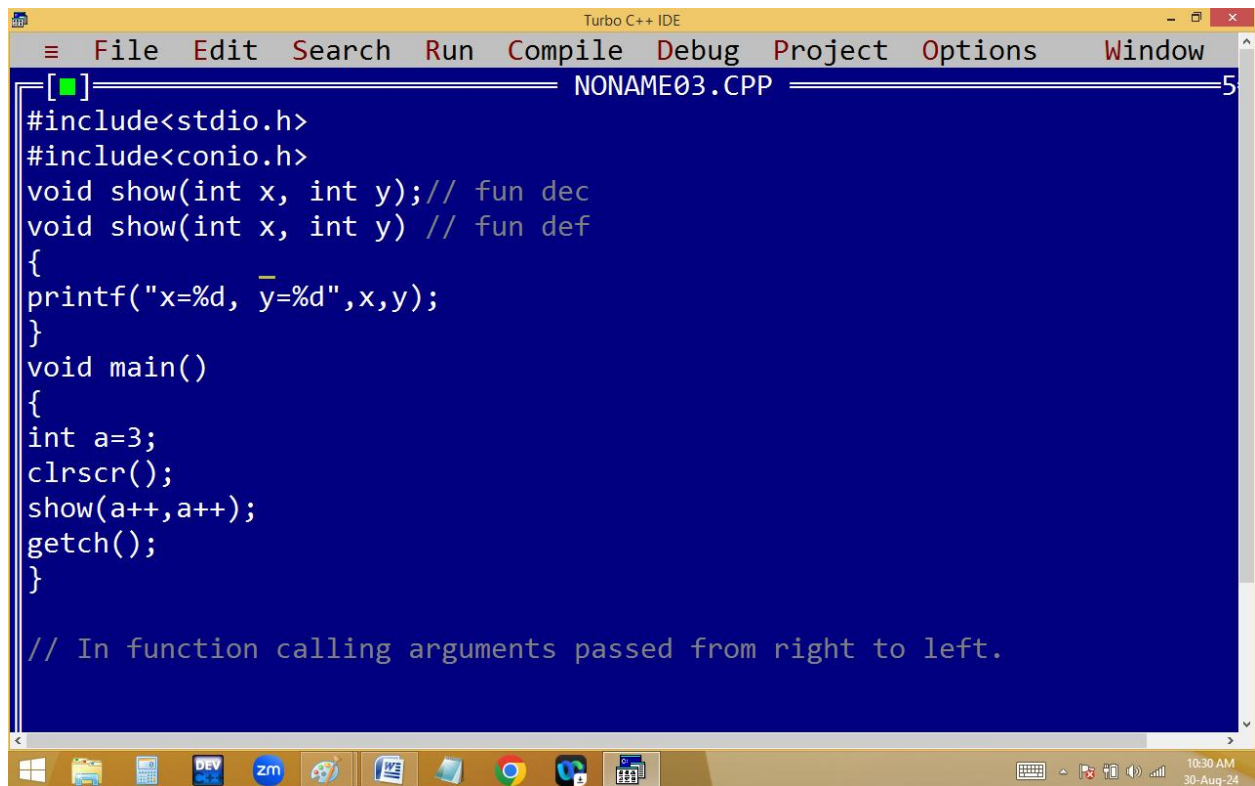
```
File Edit Search Run Compile Debug Project Options Window
[ ] NONAME03.CPP 5
#include<stdio.h>
#include<conio.h>
int evenodd(int f); // fun dec
int evenodd(int f) // fun def, f is formal parameter
{
    return f%2;
}
void main()
{
    int a;
    clrscr();
    printf("Enter a value "); scanf("%d",&a);
    if(evenodd(a)) // fun calling, a is actual parameter
        puts("Odd");
    else puts("Even");
    getch();
}
```



```
Turbo C++ IDE
Enter a value 9
Odd
```

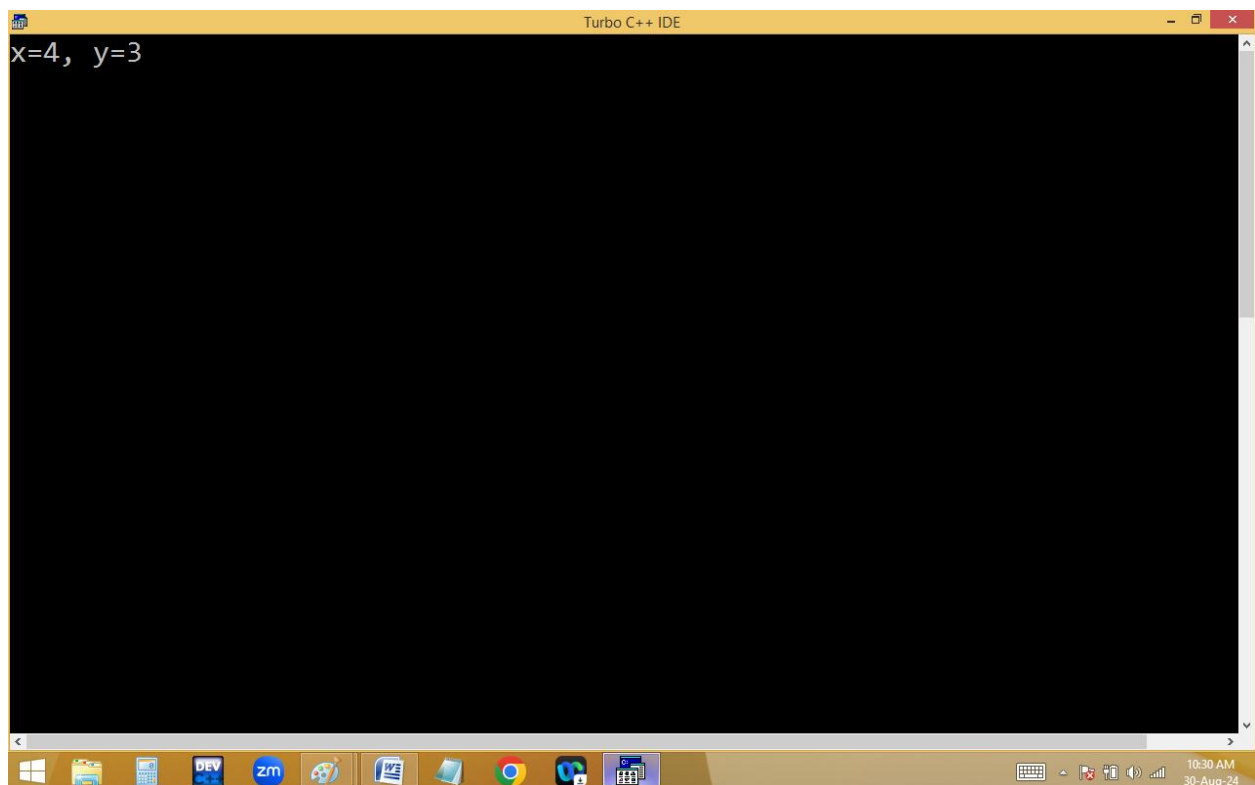


**Function with arguments, without return value:**

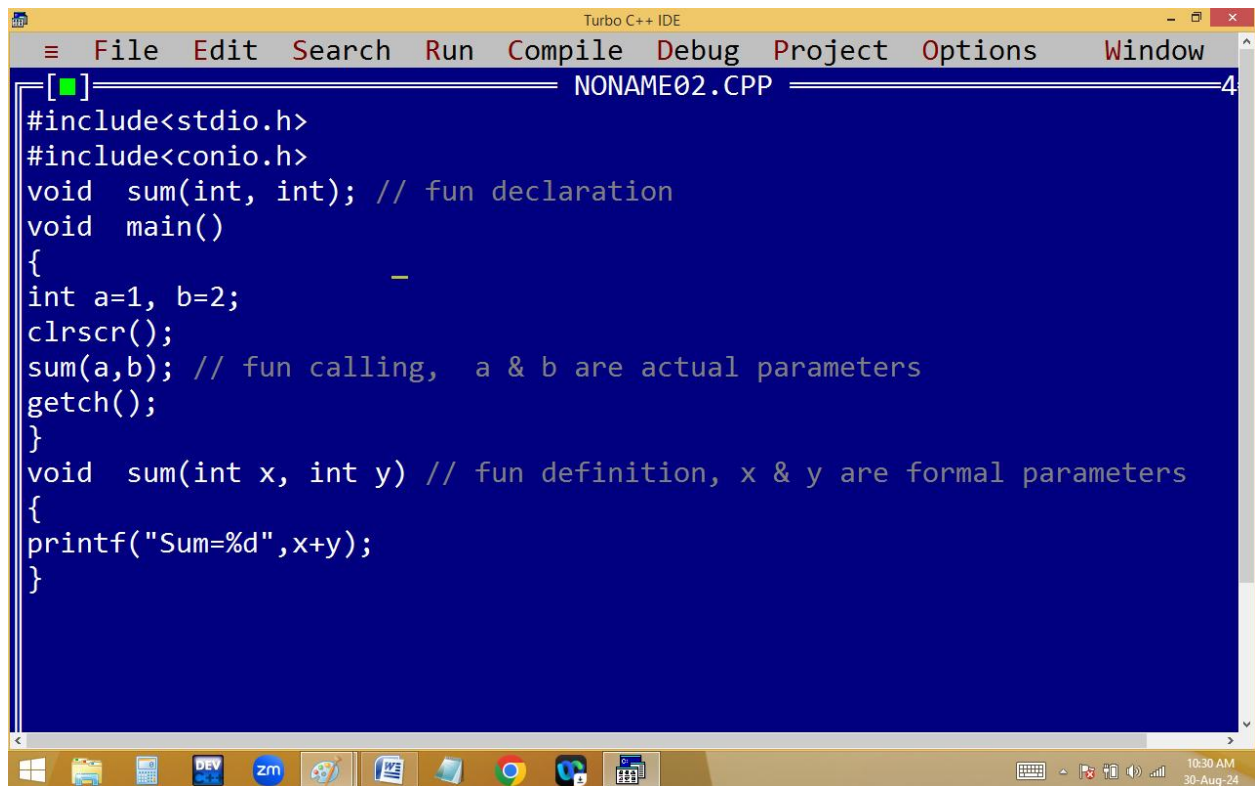


```
File Edit Search Run Compile Debug Project Options Window
[ ] NONAME03.CPP 5
#include<stdio.h>
#include<conio.h>
void show(int x, int y); // fun dec
void show(int x, int y) // fun def
{
printf("x=%d, y=%d",x,y);
}
void main()
{
int a=3;
clrscr();
show(a++,a++);
getch();
}

// In function calling arguments passed from right to left.
```

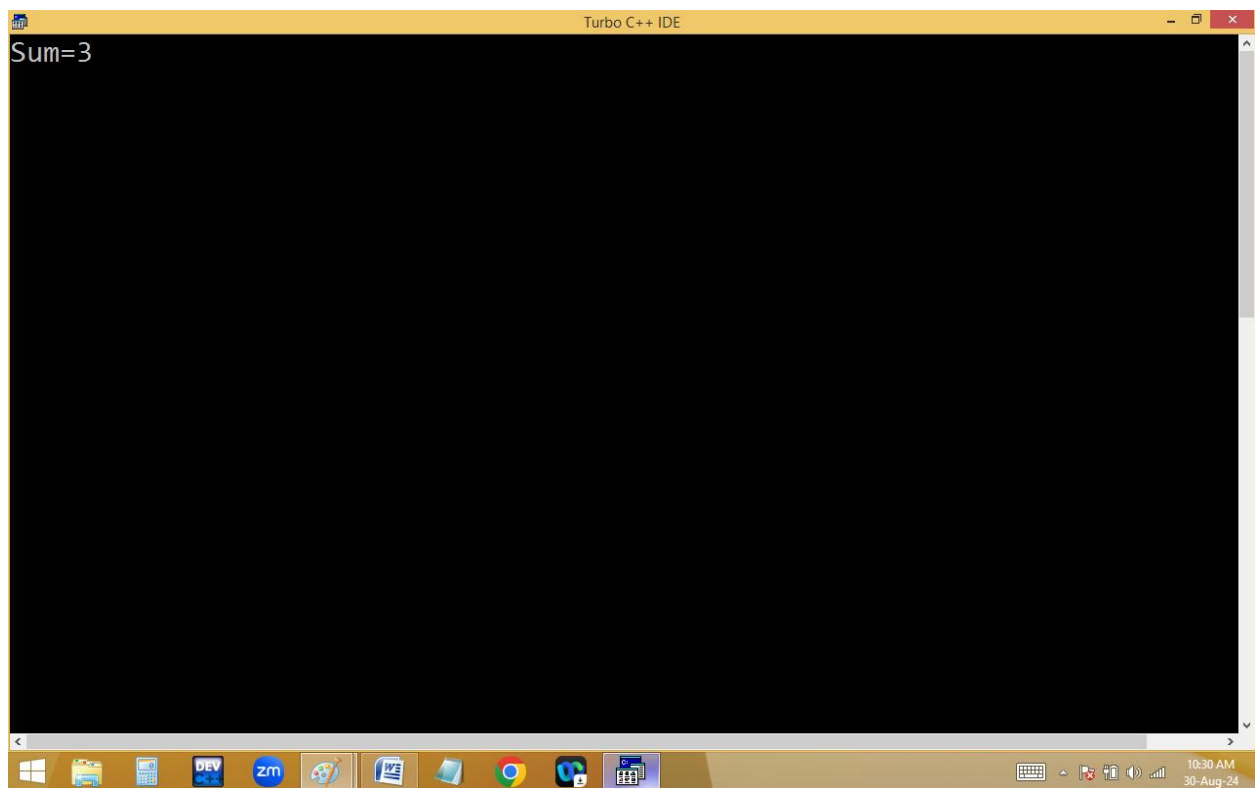


```
Turbo C++ IDE
x=4, y=3
```



The screenshot shows the Turbo C++ IDE with a menu bar (File, Edit, Search, Run, Compile, Debug, Project, Options, Window) and a toolbar. The main window displays a C++ program in a file named NONAME02.CPP. The code includes headers for stdio.h and conio.h, declares a sum function, and defines a main function that initializes variables a=1 and b=2, calls clrscr(), sum(a,b), getch(), and prints the sum. The Windows taskbar at the bottom shows various application icons and the system clock indicating 10:30 AM on 30-Aug-24.

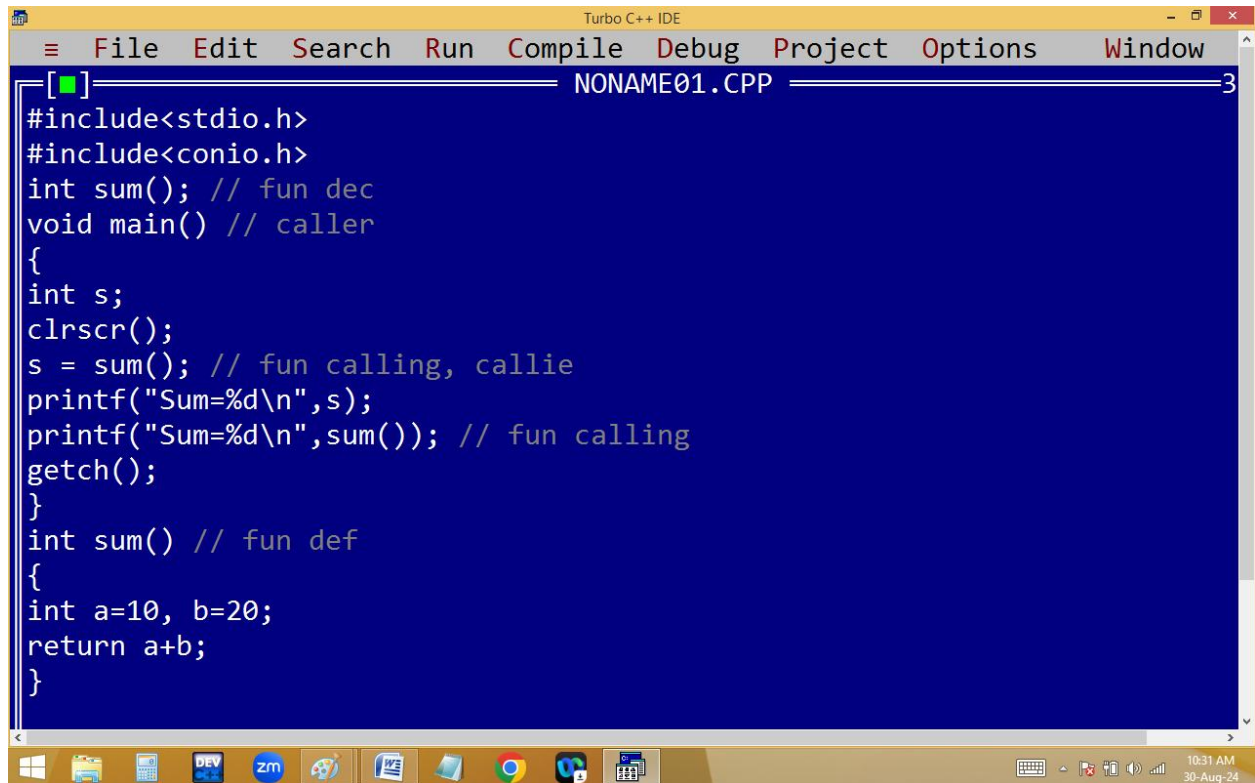
```
#include<stdio.h>
#include<conio.h>
void sum(int, int); // fun declaration
void main()
{
    int a=1, b=2;
    clrscr();
    sum(a,b); // fun calling, a & b are actual parameters
    getch();
}
void sum(int x, int y) // fun definition, x & y are formal parameters
{
    printf("Sum=%d",x+y);
}
```



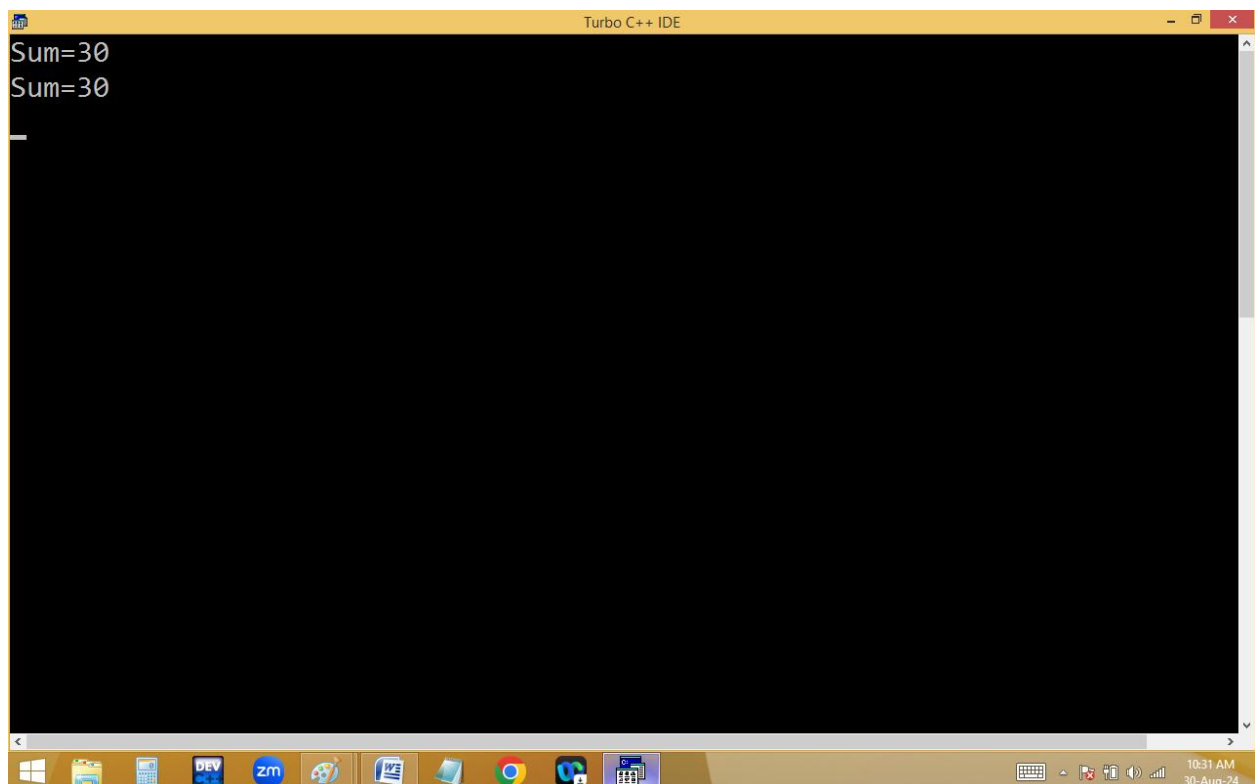
The screenshot shows the Turbo C++ IDE with the same menu bar and toolbar. The main window now displays the output of the program, which is "Sum=3". The Windows taskbar at the bottom remains the same, showing the system clock at 10:30 AM on 30-Aug-24.

```
Sum=3
```

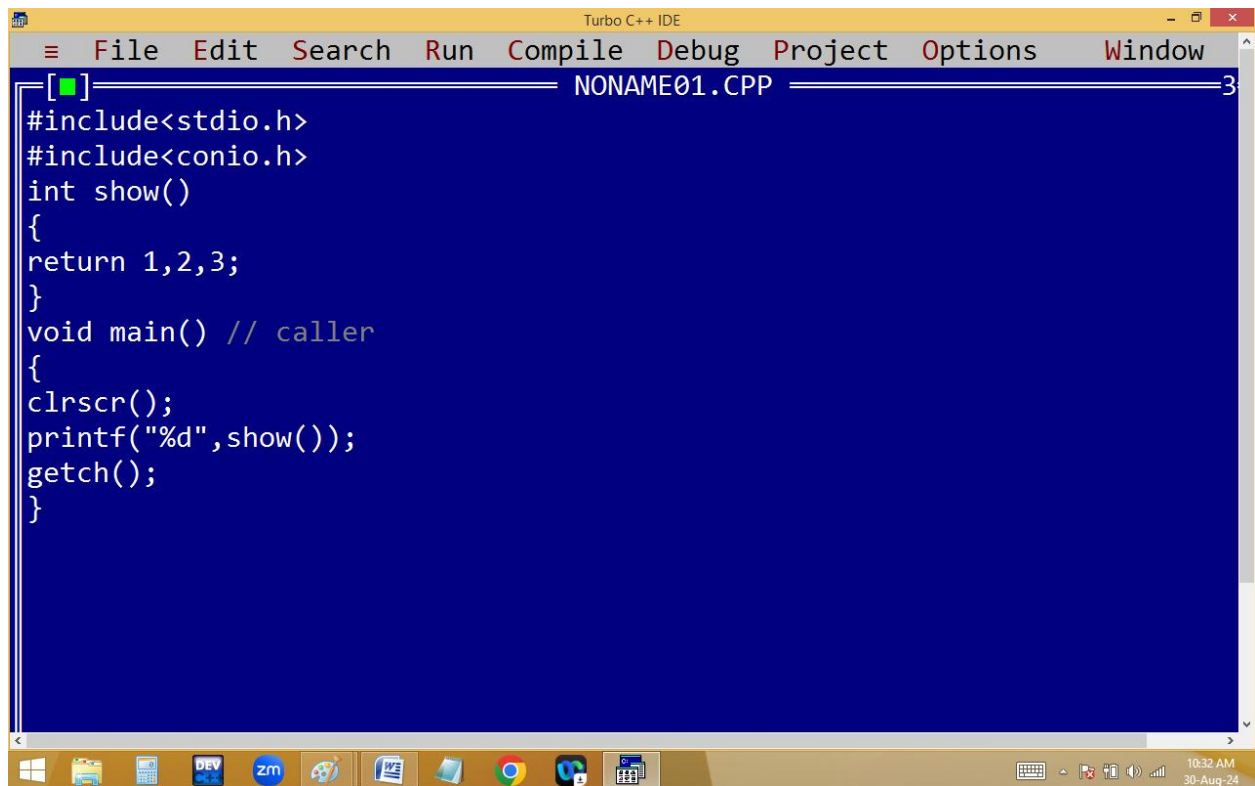
## Function without arguments, with return value:



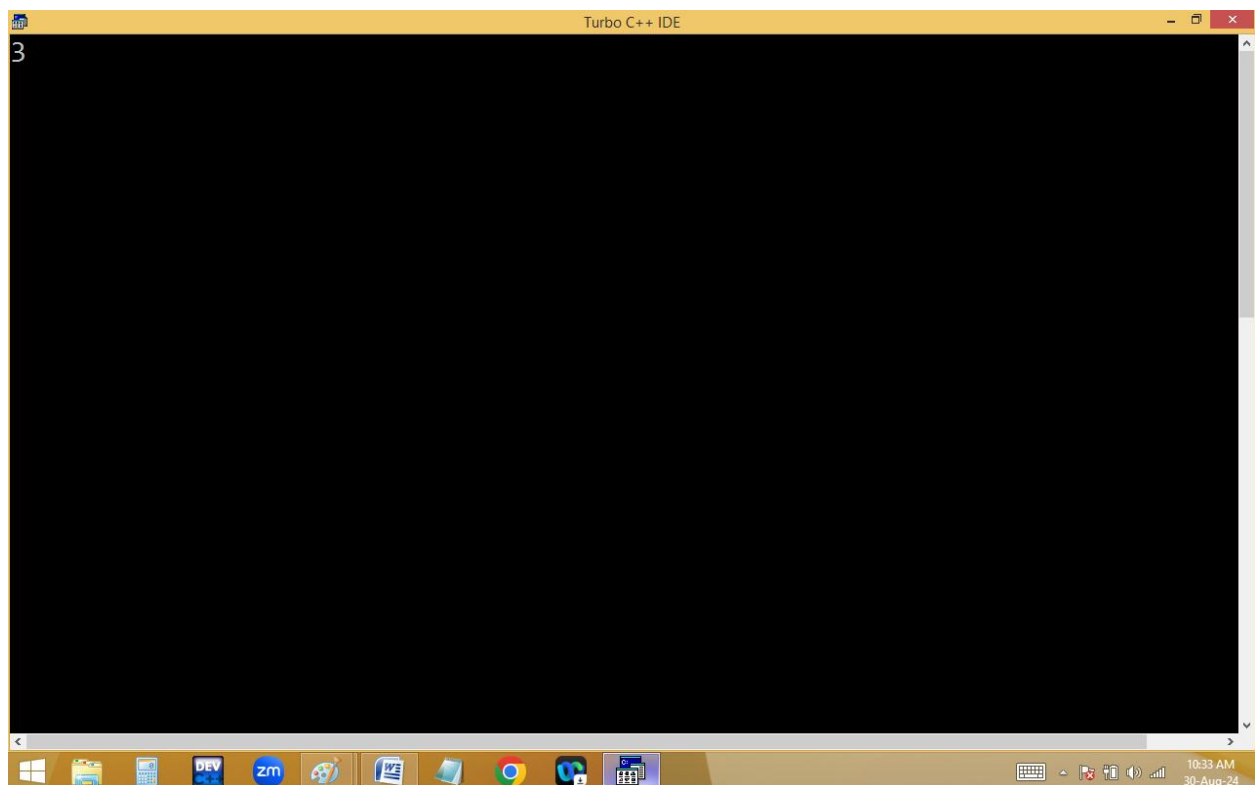
```
Turbo C++ IDE
File Edit Search Run Compile Debug Project Options Window
[ ] NONAME01.CPP 3
#include<stdio.h>
#include<conio.h>
int sum(); // fun dec
void main() // caller
{
    int s;
    clrscr();
    s = sum(); // fun calling, callie
    printf("Sum=%d\n",s);
    printf("Sum=%d\n",sum()); // fun calling
    getch();
}
int sum() // fun def
{
    int a=10, b=20;
    return a+b;
}
```

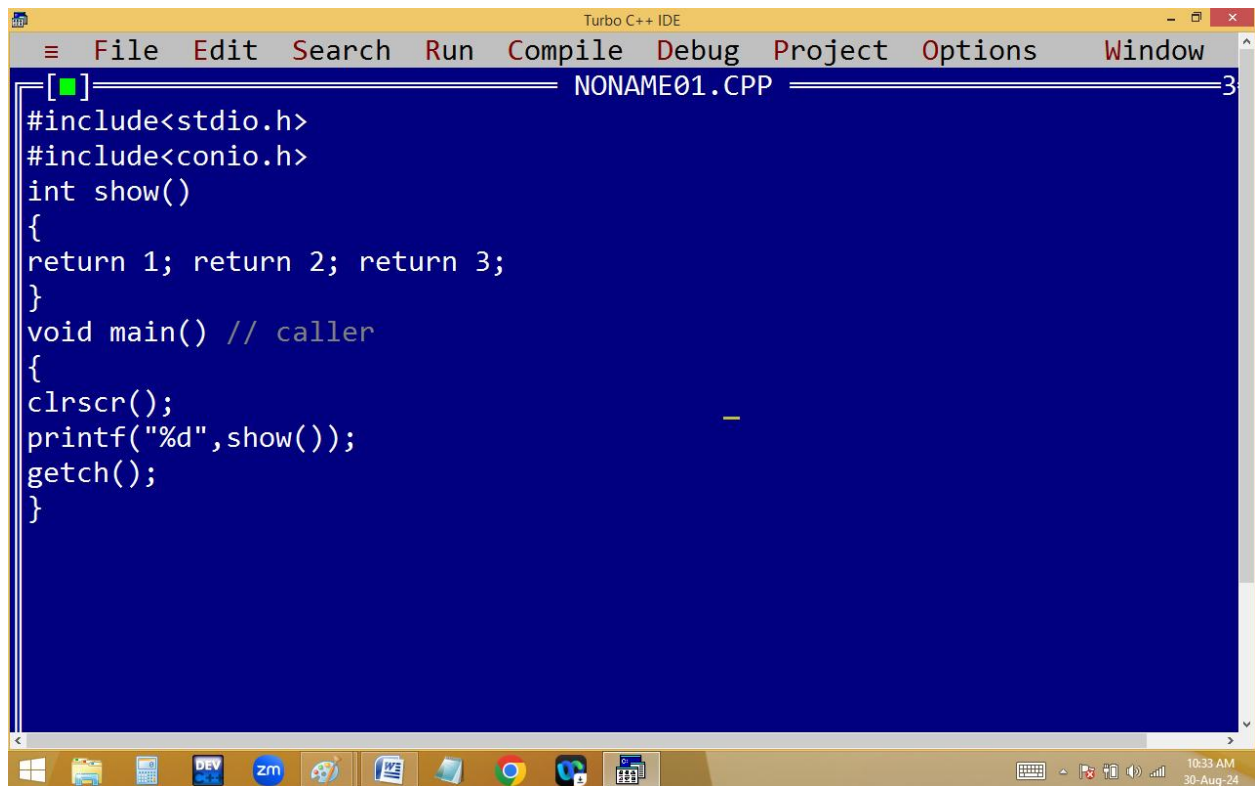


```
Turbo C++ IDE
Sum=30
Sum=30
```



```
[■] NONAME01.CPP 3
#include<stdio.h>
#include<conio.h>
int show()
{
return 1,2,3;
}
void main() // caller
{
clrscr();
printf("%d",show());
getch();
}
```





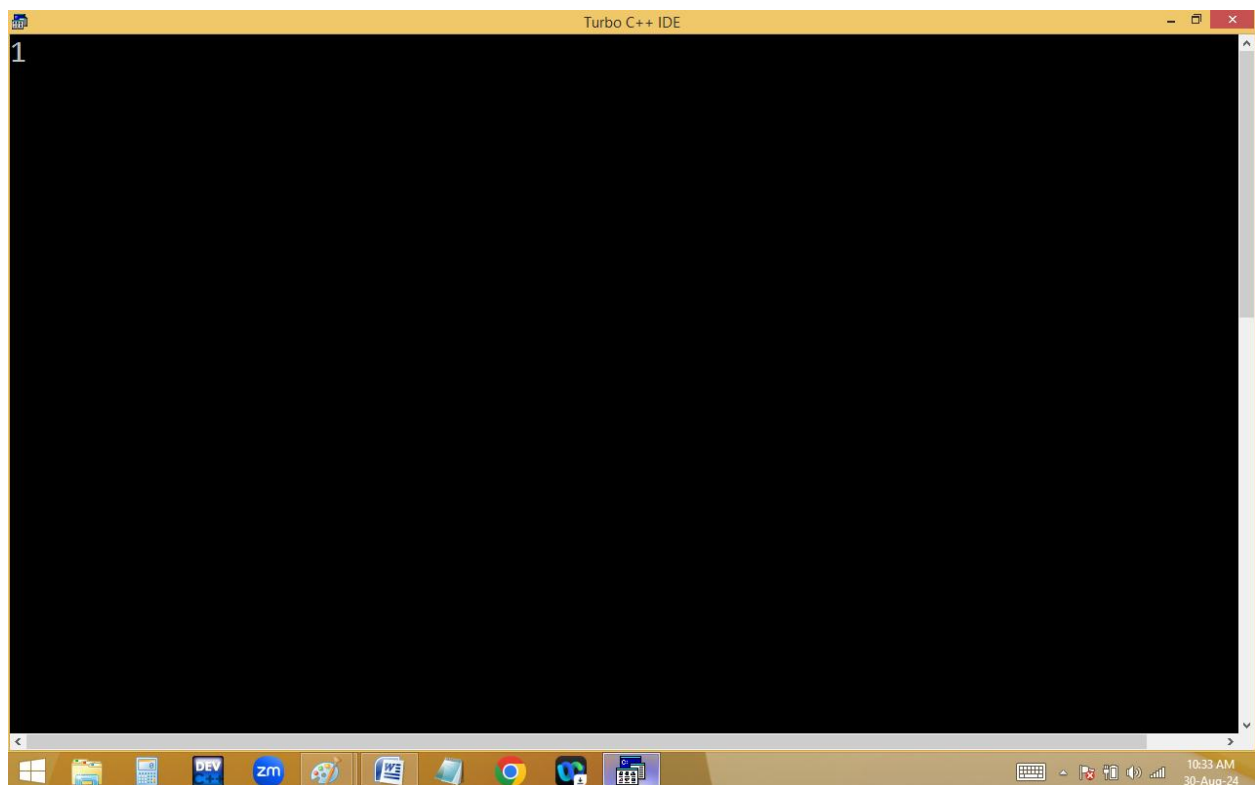
Turbo C++ IDE

File Edit Search Run Compile Debug Project Options Window

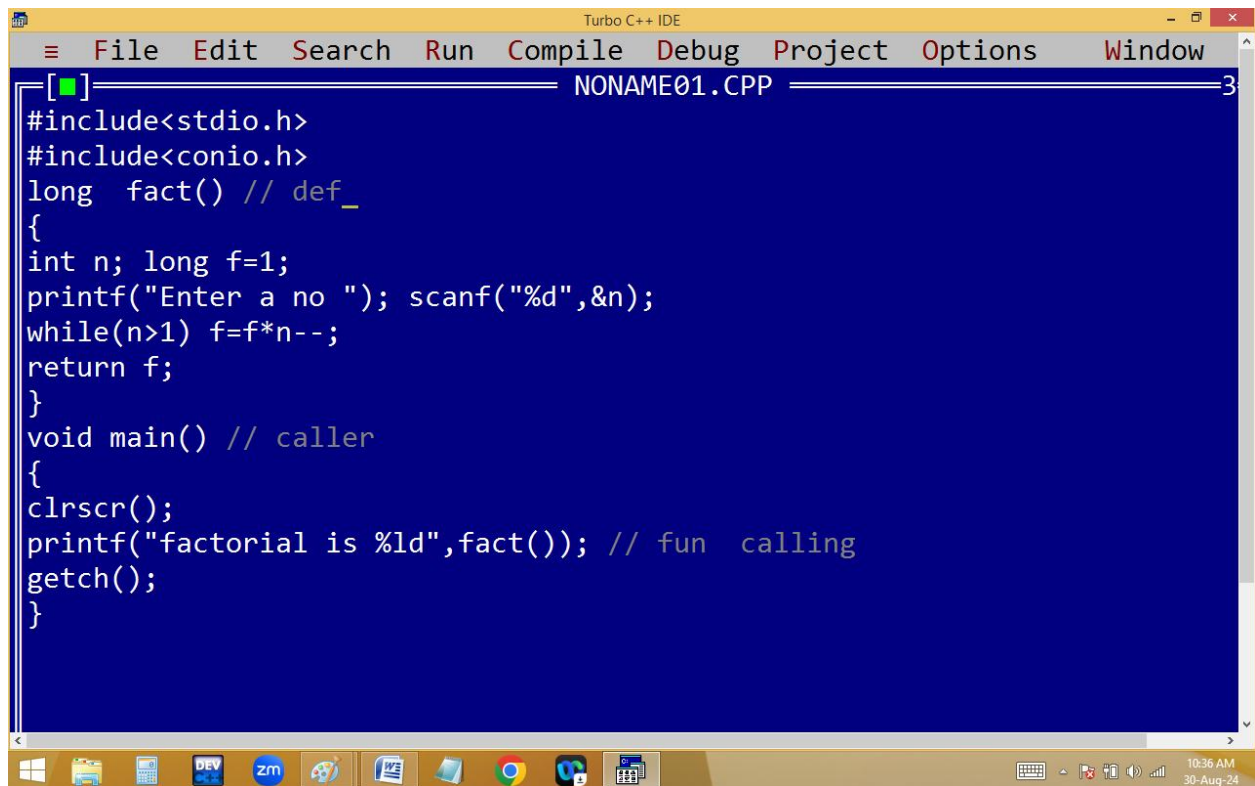
NONAME01.CPP

```
#include<stdio.h>
#include<conio.h>
int show()
{
return 1; return 2; return 3;
}
void main() // caller
{
clrscr();
printf("%d",show());
getch();
}
```

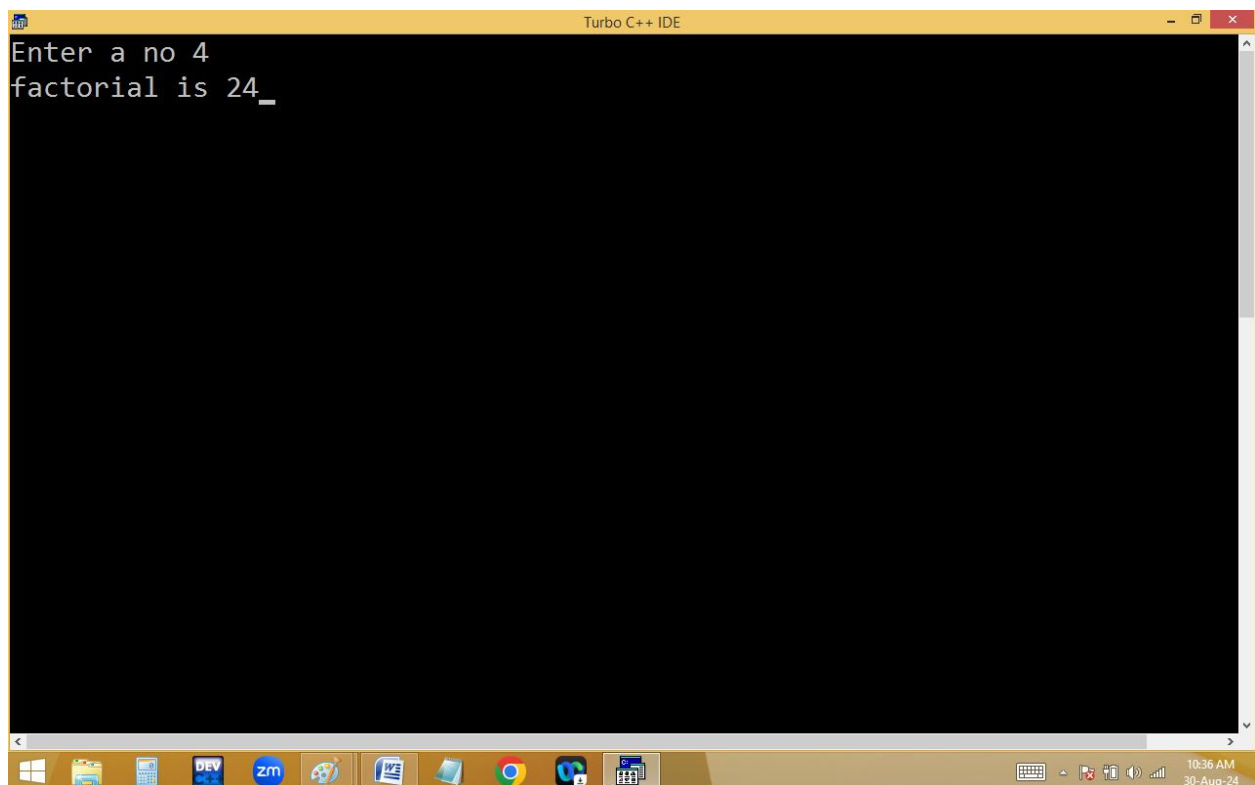
10:33 AM 30-Aug-24



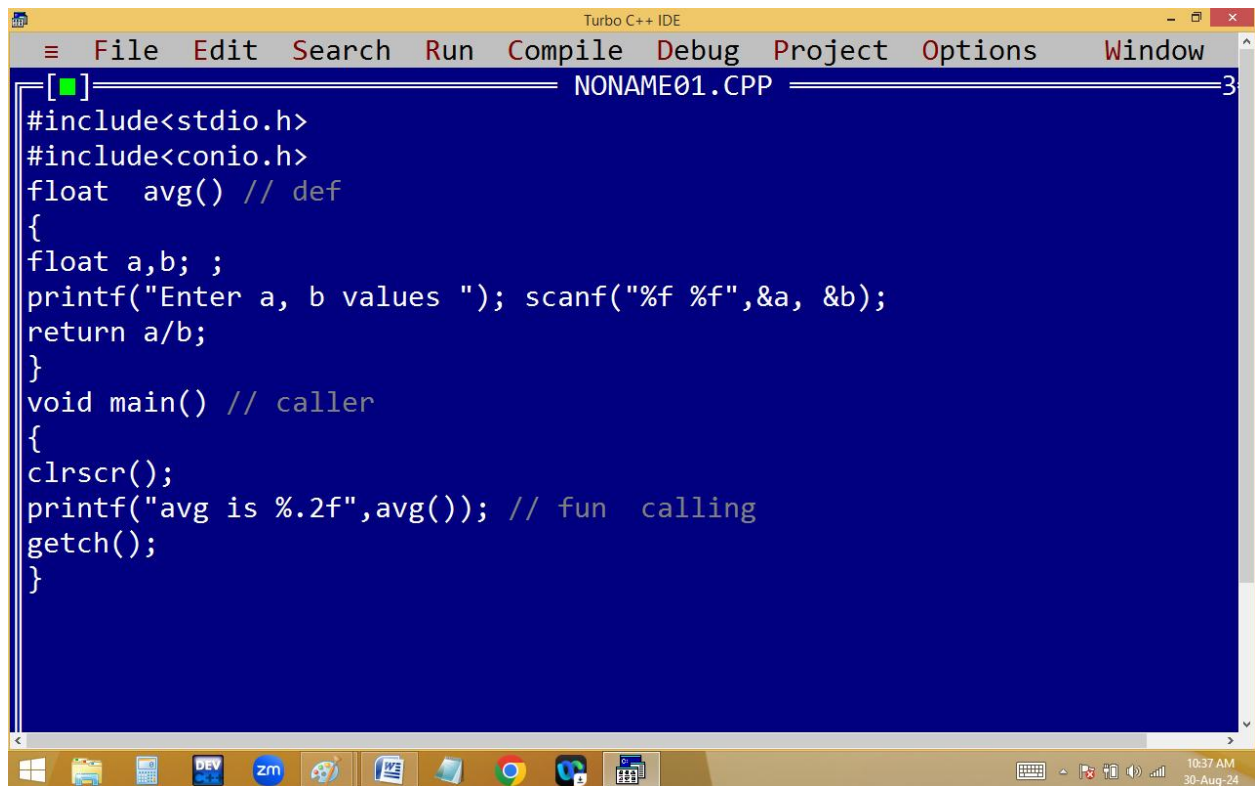




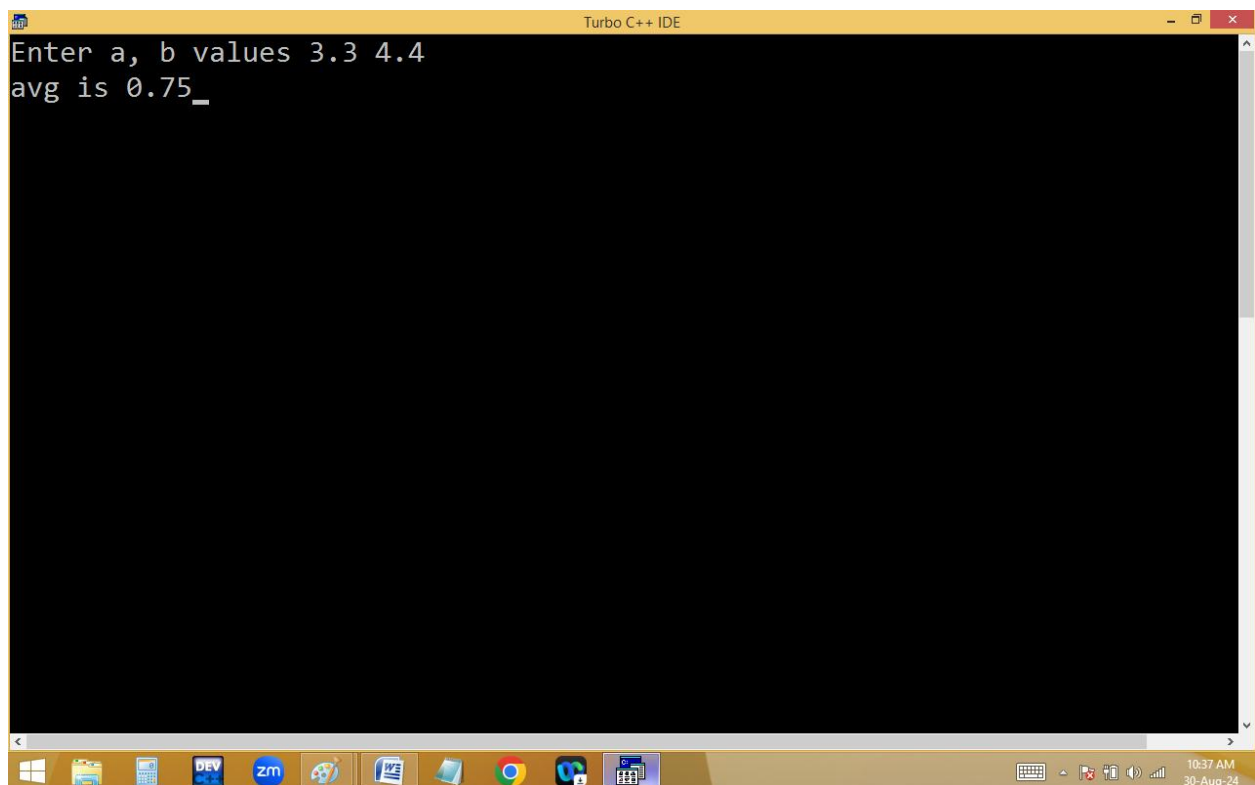
```
#include<stdio.h>
#include<conio.h>
long fact() // def_
{
int n; long f=1;
printf("Enter a no "); scanf("%d",&n);
while(n>1) f=f*n--;
return f;
}
void main() // caller
{
clrscr();
printf("factorial is %ld",fact()); // fun calling
getch();
}
```



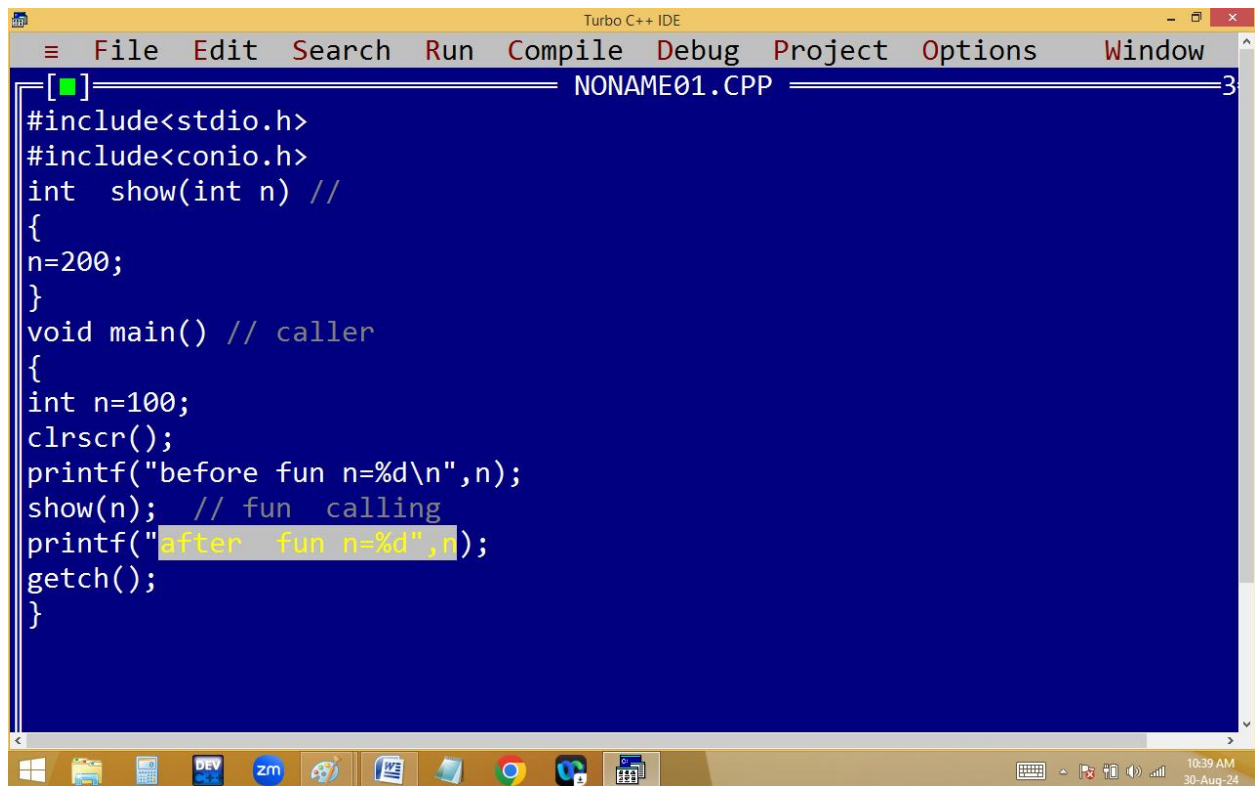
```
Enter a no 4
factorial is 24_
```



```
File Edit Search Run Compile Debug Project Options Window
[ ] NONAME01.CPP 3
#include<stdio.h>
#include<conio.h>
float avg() // def
{
float a,b; ;
printf("Enter a, b values "); scanf("%f %f",&a, &b);
return a/b;
}
void main() // caller
{
clrscr();
printf("avg is %.2f",avg()); // fun calling
getch();
}
```

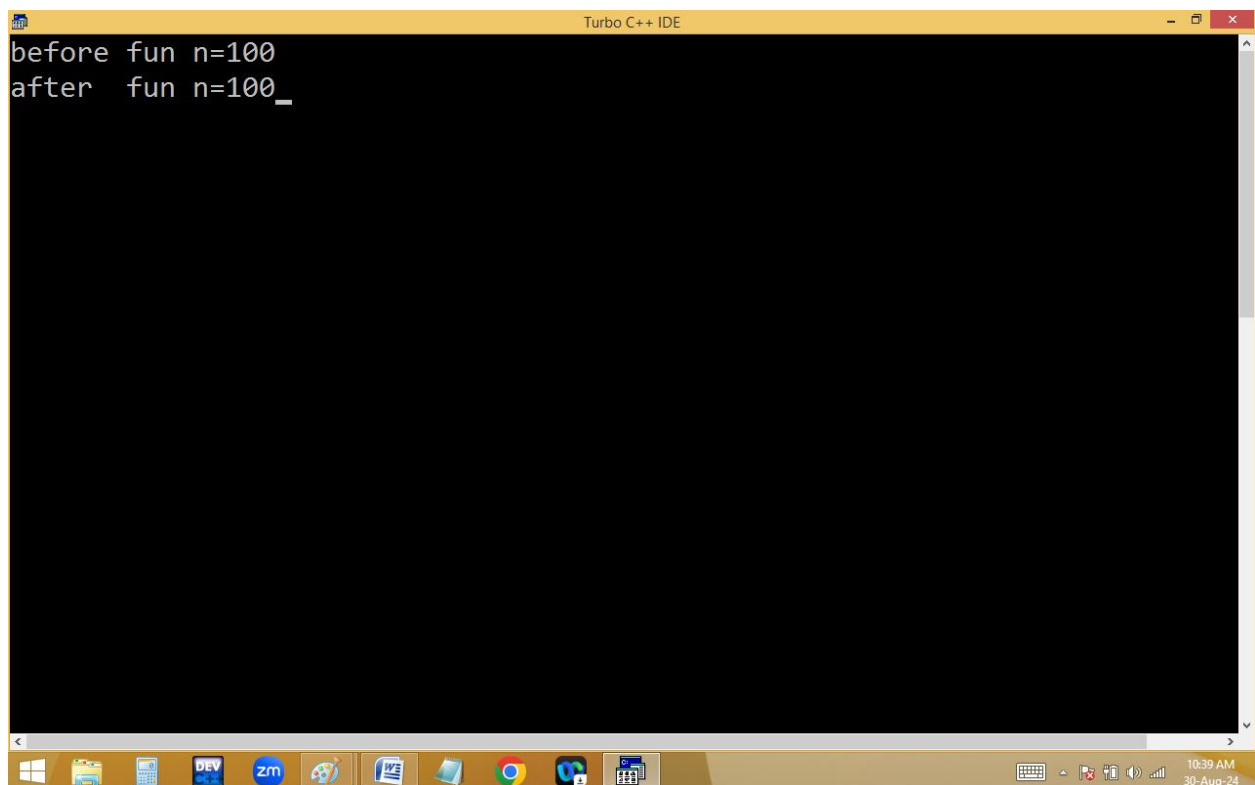


```
Enter a, b values 3.3 4.4
avg is 0.75_
```

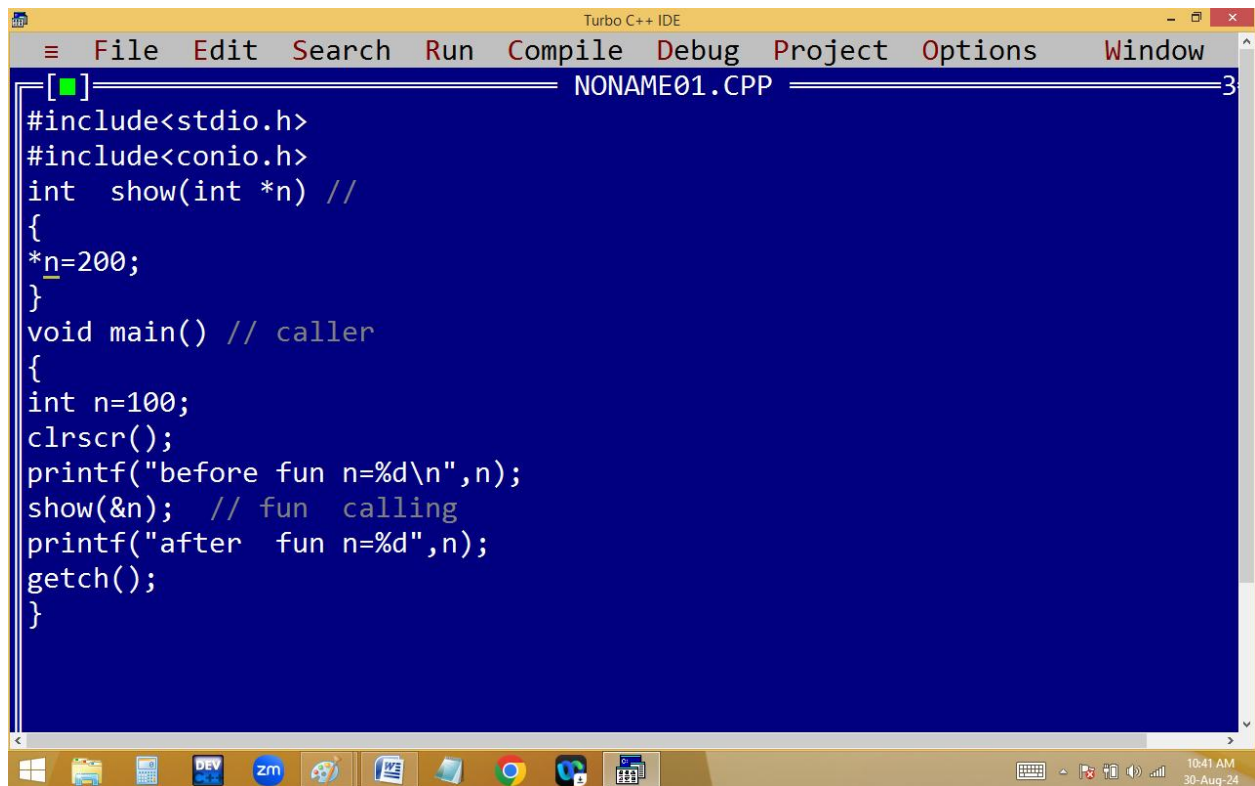


```
File Edit Search Run Compile Debug Project Options Window
NONAME01.CPP
#include<stdio.h>
#include<conio.h>
int show(int n) //
{
n=200;
}
void main() // caller
{
int n=100;
clrscr();
printf("before fun n=%d\n",n);
show(n); // fun calling
printf("after fun n=%d",n);
getch();
}
```

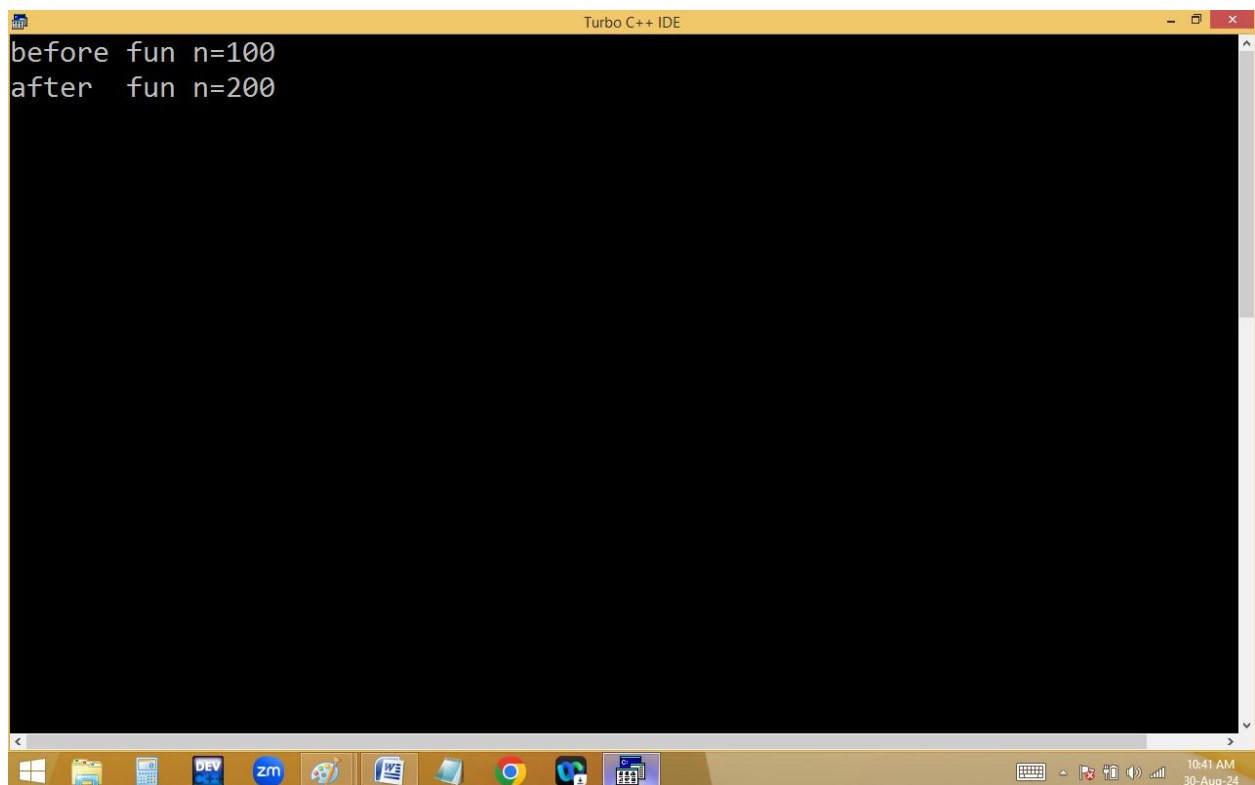
before fun n=100  
after fun n=100\_



```
before fun n=100
after fun n=100_
```

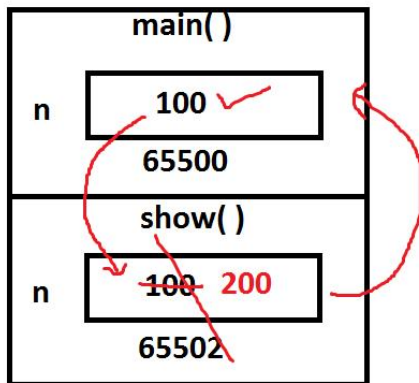


```
File Edit Search Run Compile Debug Project Options Window
NONAME01.CPP
#include<stdio.h>
#include<conio.h>
int show(int *n) //
{
    *n=200;
}
void main() // caller
{
    int n=100;
    clrscr();
    printf("before fun n=%d\n",n);
    show(&n); // fun calling
    printf("after fun n=%d",n);
    getch();
}
```



```
before fun n=100
after fun n=200
```

**stack  
call by value**



**call by address/reference**

