F.Y.B.Sc.IT-SEM 1

Programming Principles With C (PUSIT101)

By,
Prof. Nikita Madwal

Unit III

4. Functions and Program Structure

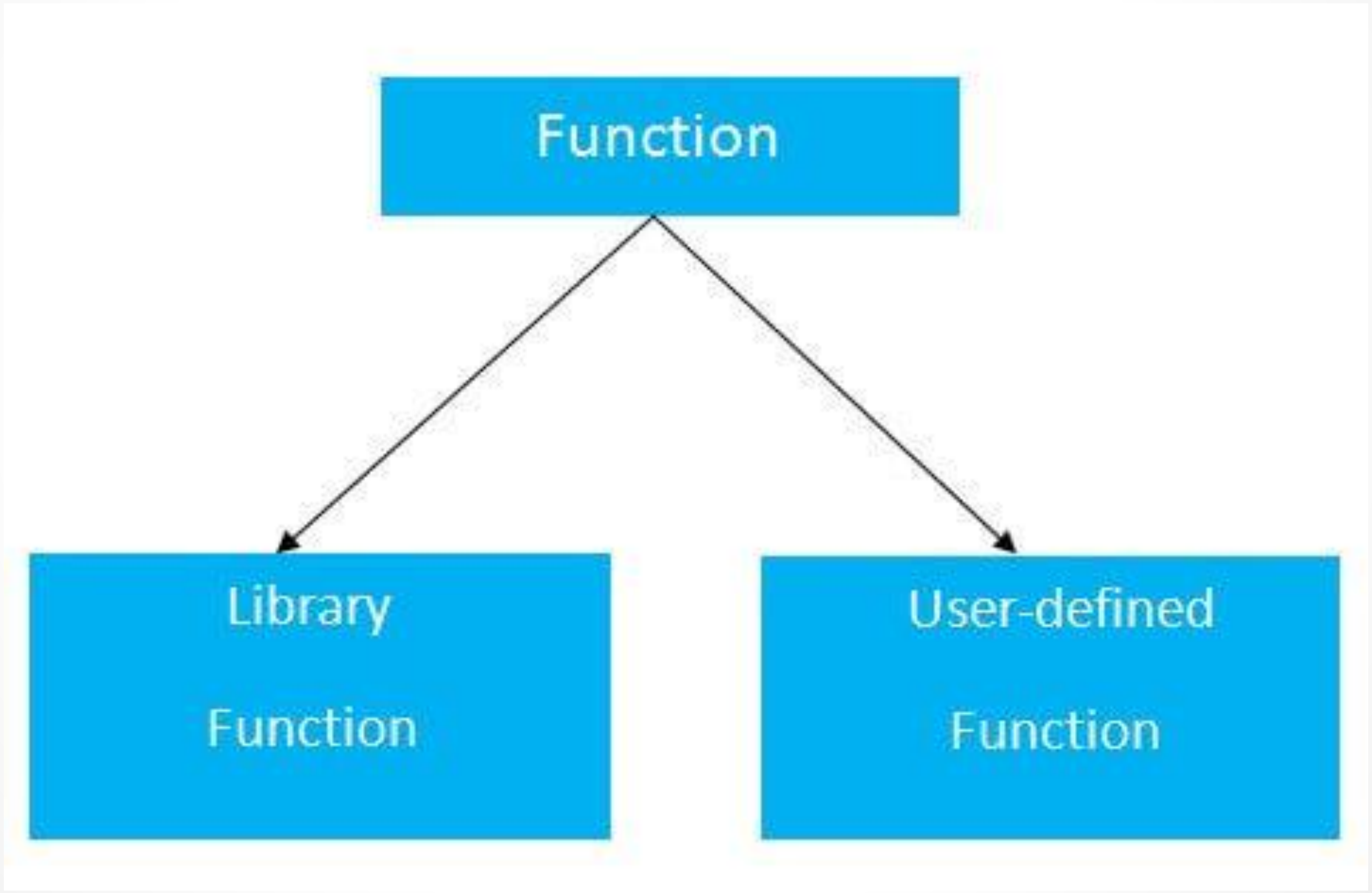
Basics of functions

- C functions can be viewed as basic building blocks of the program.
- A function is a **self-contained program** segment that carries out **some specific**, **well-defined task**.
- A function is a group of statements that together perform a task.
- Every C program consists of one or more functions
- Every C program has **at least one function**, which is **main()**,
- Additional functions will be subordinate to main
- Essence of modular programming lies in basic building blocks of the program. These blocks are known as **C function**.
- C functions comprises of set of instructions delimited inside by **{ }** braces

- **Advantage of functions in C**

- By using functions, we can **avoid rewriting same logic/code again and again** in a program.
- These functions can be **called multiple times** as and when required, there is no limit in calling C functions.
- These functions can be **called from any part of the program** (from any place in a program)
- By Using function it is easy to keep a track of large C program easily when it is divided into multiple functions.
- It reduces the complexity of a big program and optimizes the code.

Types of function



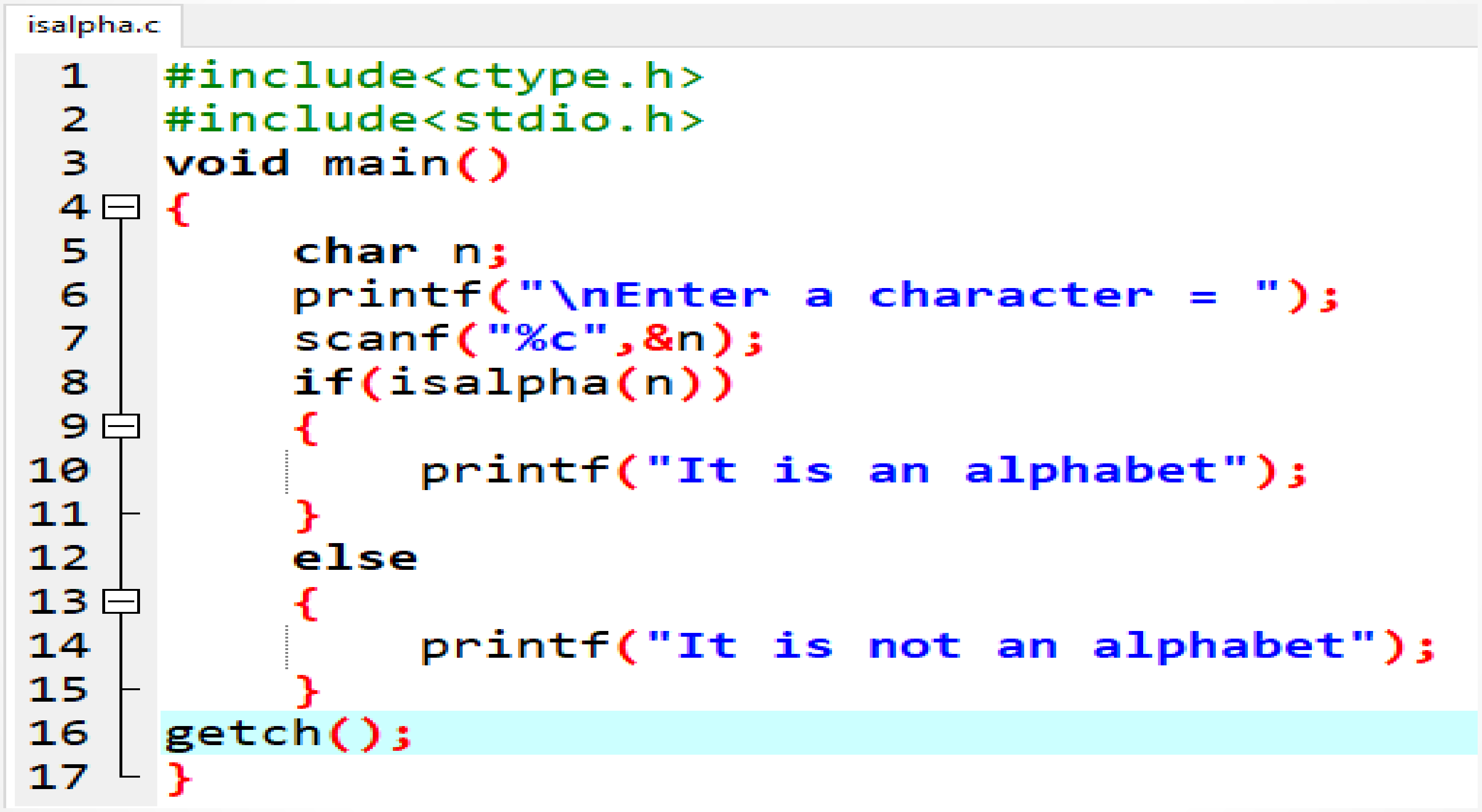
- **Library Functions**
- Library functions are ready made or they are **built-in functions** made available through C library
- These are the functions which are declared in the C header files
- For e.g. scanf(), printf()
- **User-defined function**
- These functions are those functions which are defined by the time of writing of program.
- The functions which are **created by the C programmer** is known as **User-defined function**
- so that he/she can use it many times.

Library Function

(Character Functions)

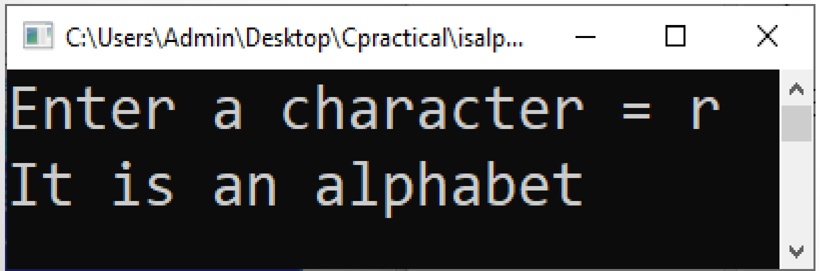
❖ **The header file required for character function is<ctype.h>**

➢ **isalpha() -** checks whether character is alphabetic



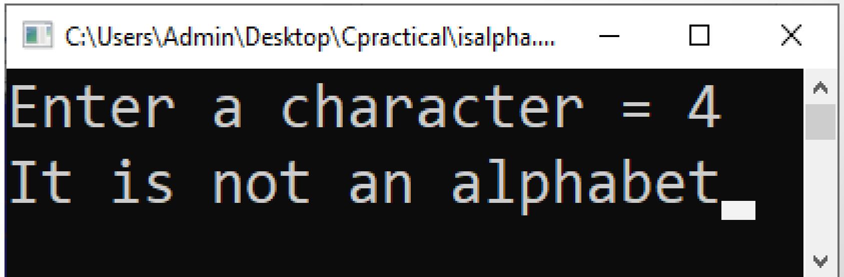
isalpha.c

```c
#include<ctype.h>
#include<stdio.h>
void main()
{
	char n;
	printf("\nEnter a character = ");
	scanf("%c",&n);
	if(isalpha(n))
	{
		printf("It is an alphabet");
	}
	else
	{
		printf("It is not an alphabet");
	}
getch();
}
```



C:\Users\Admin\Desktop\Cpractical\isalp...

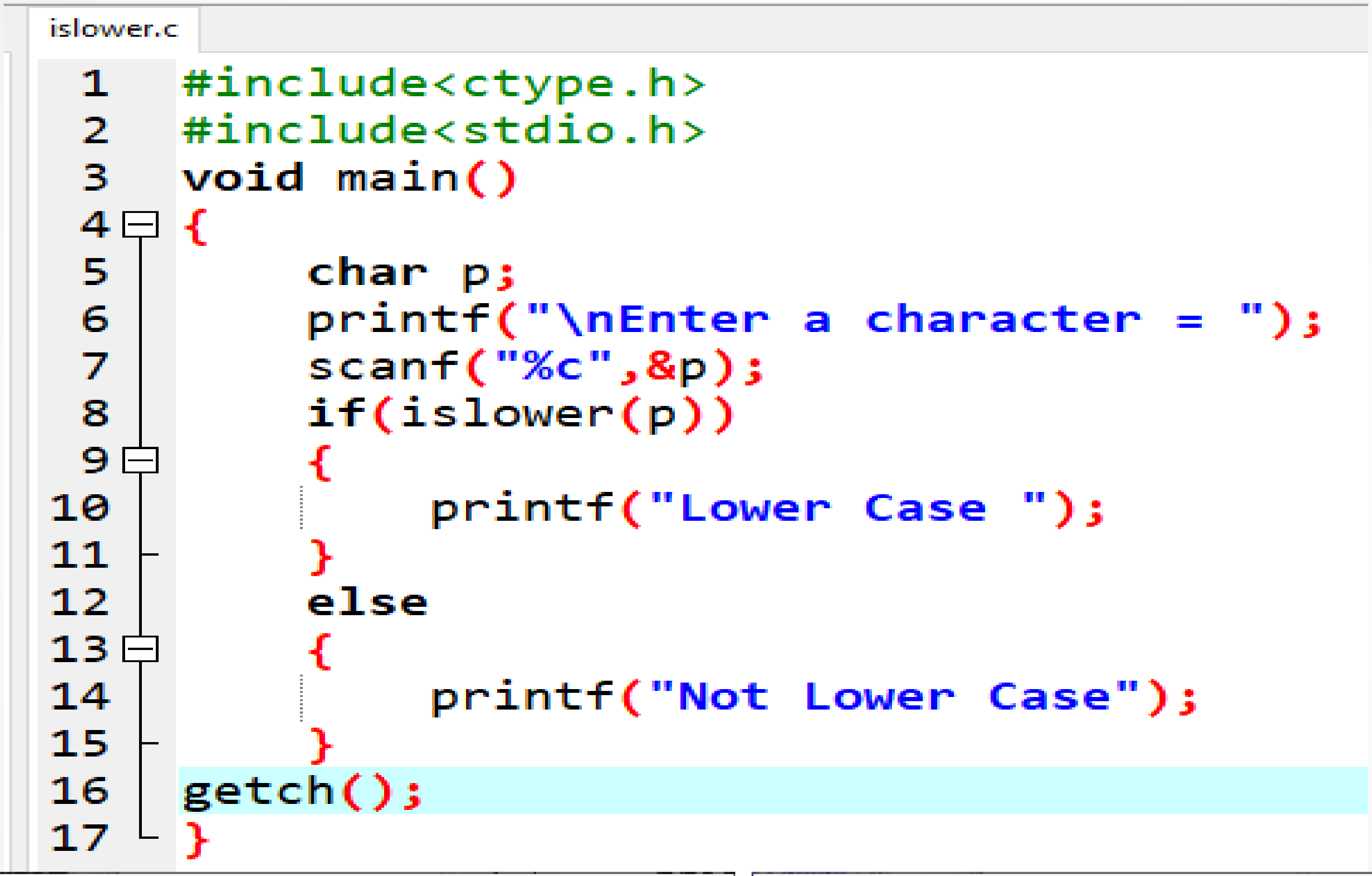
```
Enter a character = r
It is an alphabet
```



C:\Users\Admin\Desktop\Cpractical\isalpha....

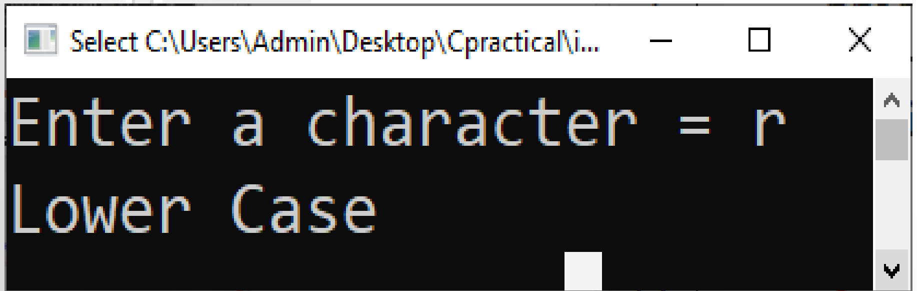
```
Enter a character = 4
It is not an alphabet_
```

- **islower() -** Checks whether character is lower case
- **isupper() -** Checks whether character is upper case



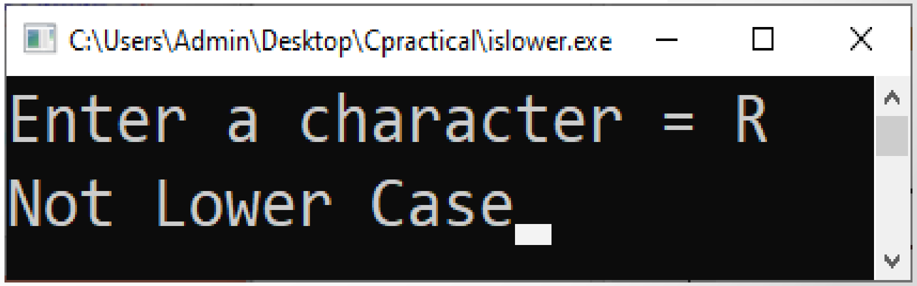
islower.c

```c
#include<ctype.h>
#include<stdio.h>
void main()
{
	char p;
	printf("\nEnter a character = ");
	scanf("%c",&p);
	if(islower(p))
	{
		printf("Lower Case ");
	}
	else
	{
		printf("Not Lower Case");
	}
getch();
}
```



Select C:\Users\Admin\Desktop\Cpractical\i...

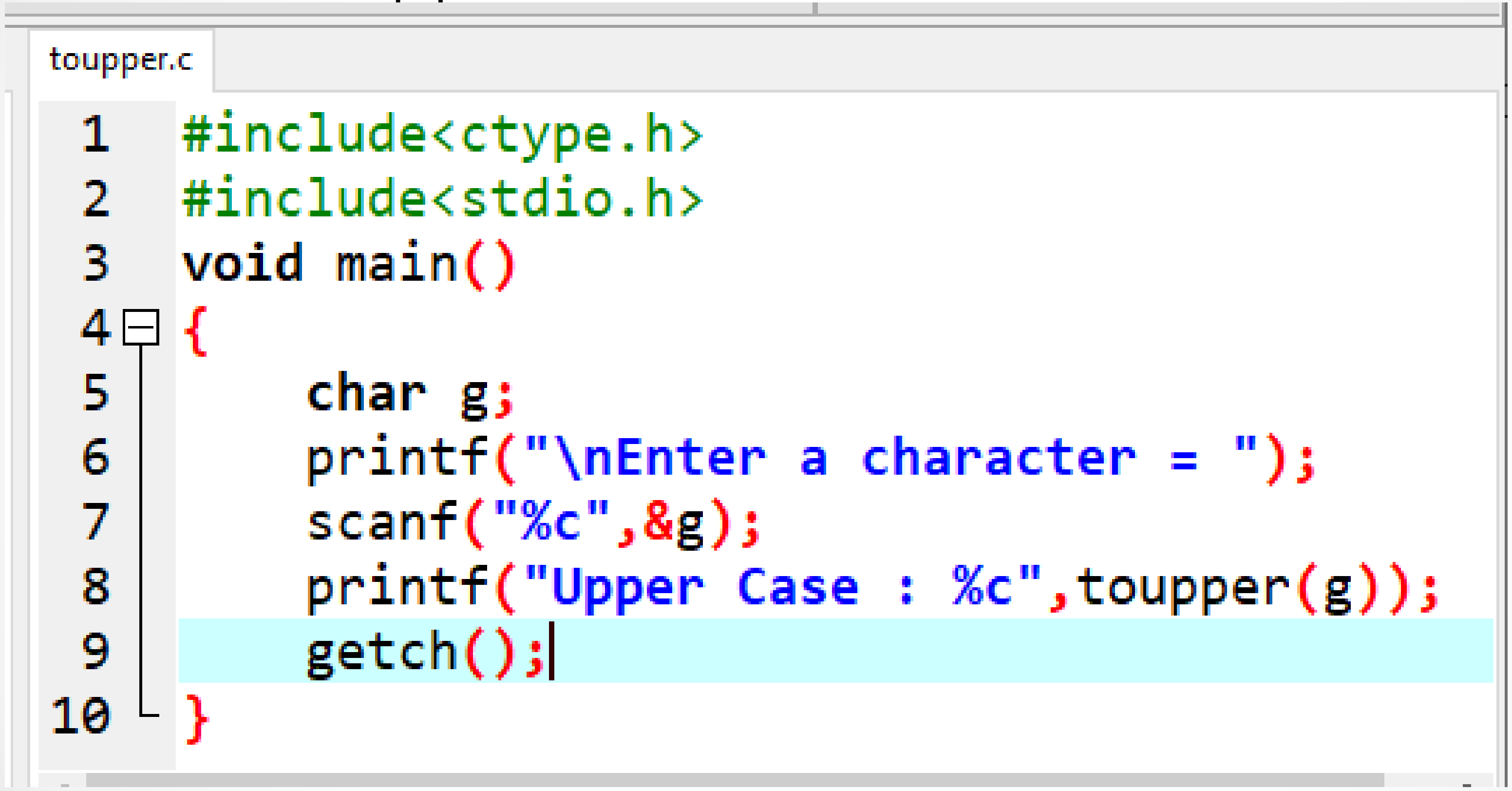
```
Enter a character = r
Lower Case
```



C:\Users\Admin\Desktop\Cpractical\islower.exe

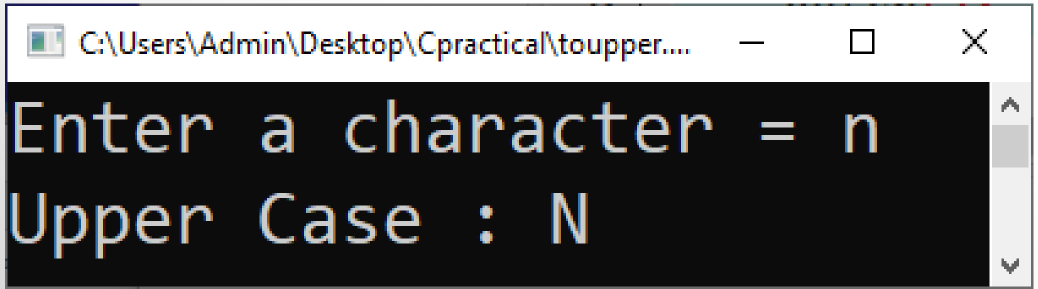
```
Enter a character = R
Not Lower Case
```

- **tolower() -** Checks whether character is alphabetic & converts to lower case
- **toupper() -** Checks whether character is alphabetic & converts to upper case



toupper.c

```c
#include<ctype.h>
#include<stdio.h>
void main()
{
	char g;
	printf("\nEnter a character = ");
	scanf("%c",&g);
	printf("Upper Case : %c",toupper(g));
	getch();
}
```



C:\Users\Admin\Desktop\Cpractical\toupper....

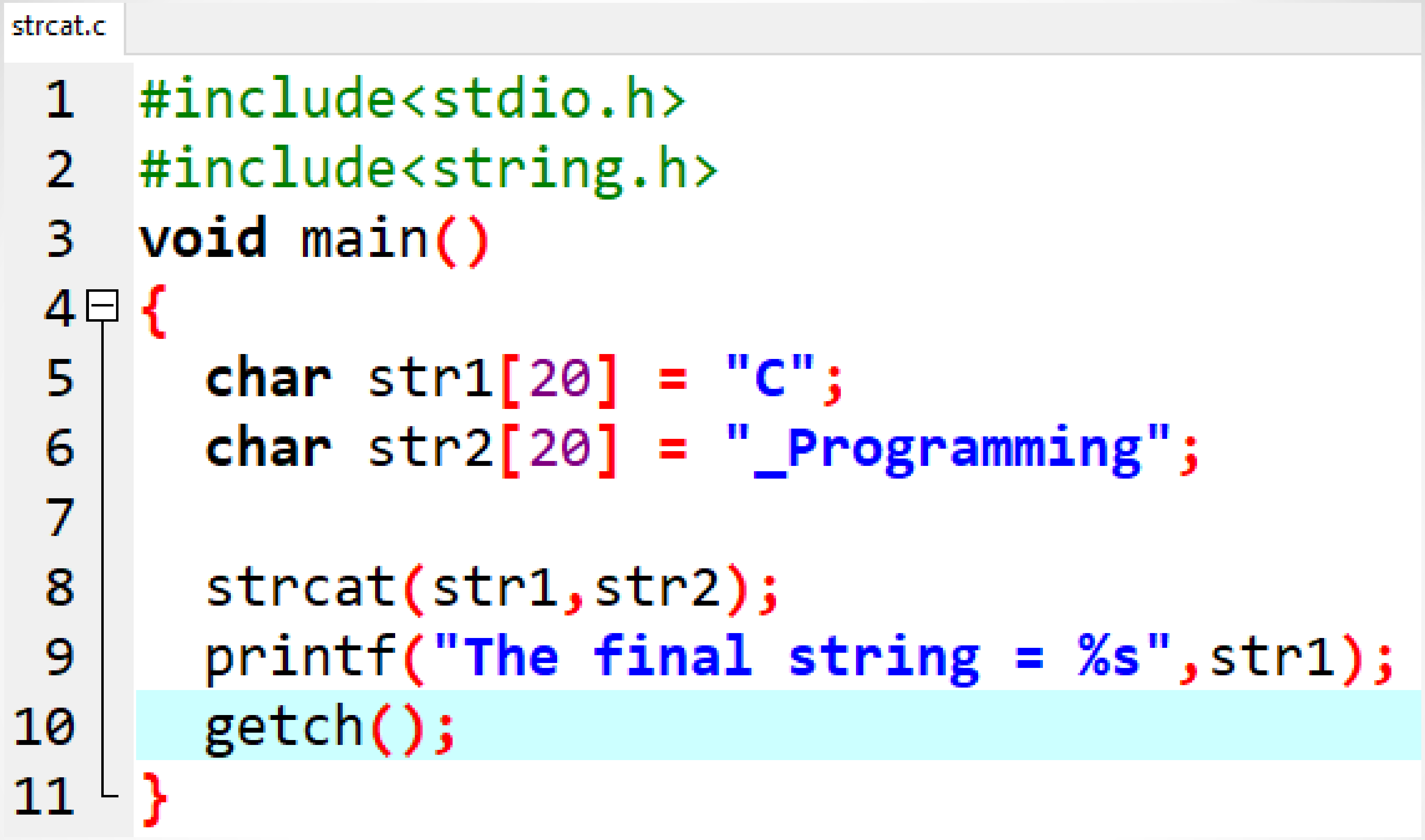
```
Enter a character = n
Upper Case : N
```

(String Functions)

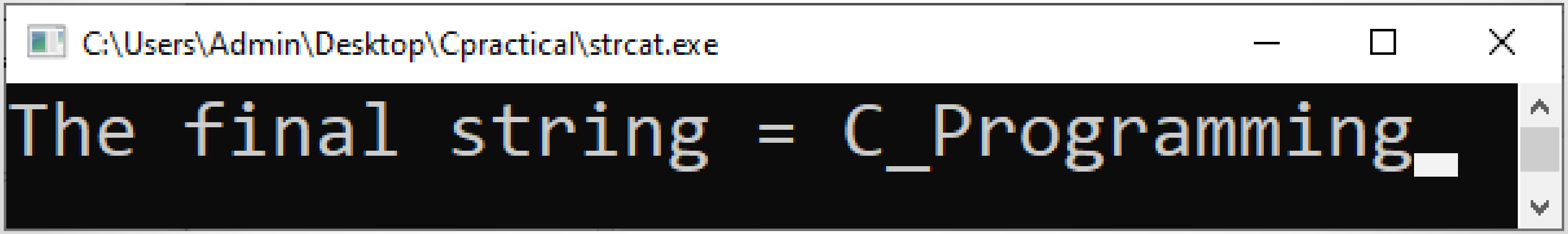
❖ **The header file required for string function is <string.h>**

➢ **strcat()**

- This function in C language concatenates two given strings.
- It concatenates source string at the end of destination string.
- Example:
  strcat (str2, str1); – str1 is concatenated at the end of str2.
  strcat (str1, str2); – str2 is concatenated at the end of str1.


strcat.c

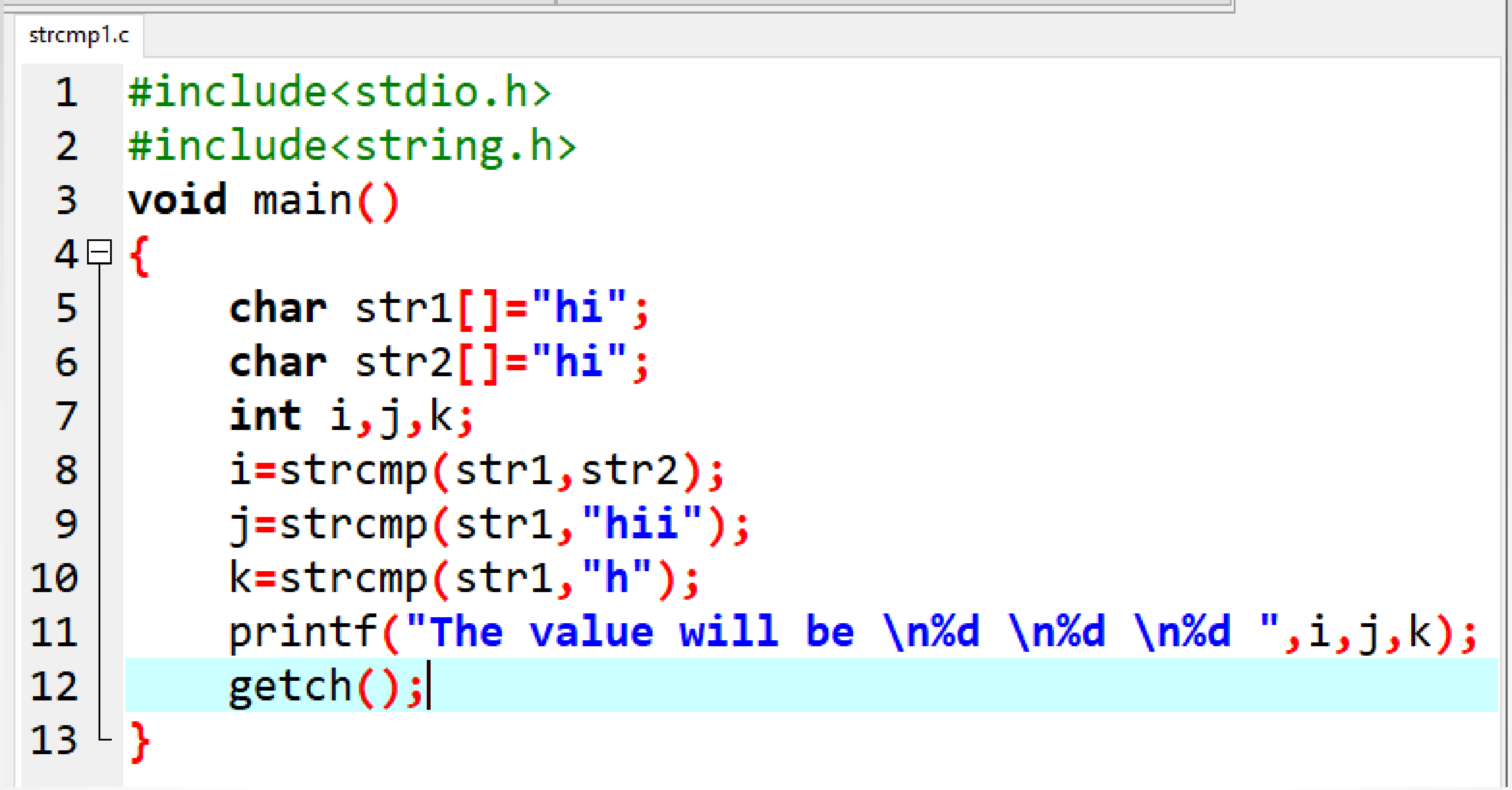
```c
#include<stdio.h>
#include<string.h>
void main()
{
   char str1[20] = "C";
   char str2[20] = "_Programming";

   strcat(str1,str2);
   printf("The final string = %s",str1);
   getch();
}
```


C:\Users\Admin\Desktop\Cpractical\strcat.exe

```
The final string = C_Programming
```

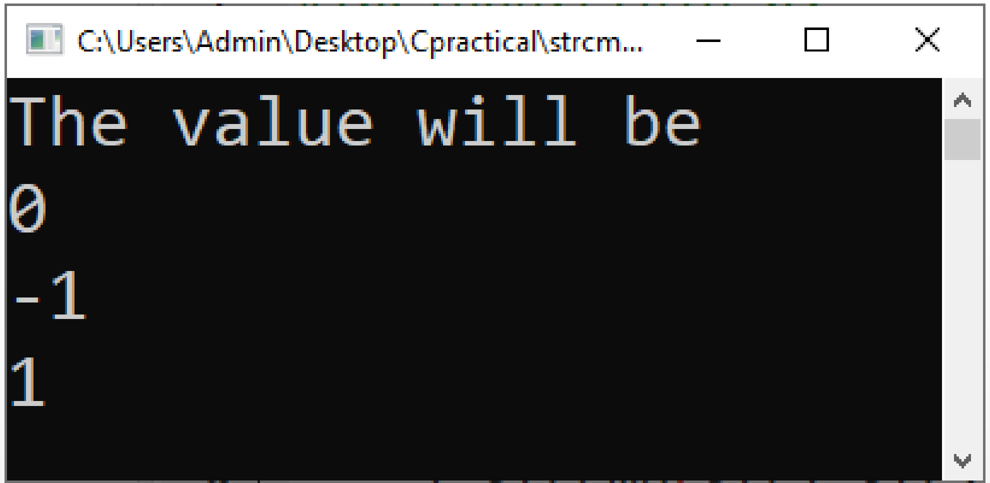
- **strcmp()**

- strcmp( ) function in C compares two given strings
- Returns zero if they are same.
- If length of string1 < string2, it returns < 0 value.
- If length of string1 > string2, it returns > 0 value.



strcmp1.c

```c
#include<stdio.h>
#include<string.h>
void main()
{
	char str1[]="hi";
	char str2[]="hi";
	int i,j,k;
	i=strcmp(str1,str2);
	j=strcmp(str1,"hii");
	k=strcmp(str1,"h");
	printf("The value will be \n%d \n%d \n%d ",i,j,k);
	getch();
}
```

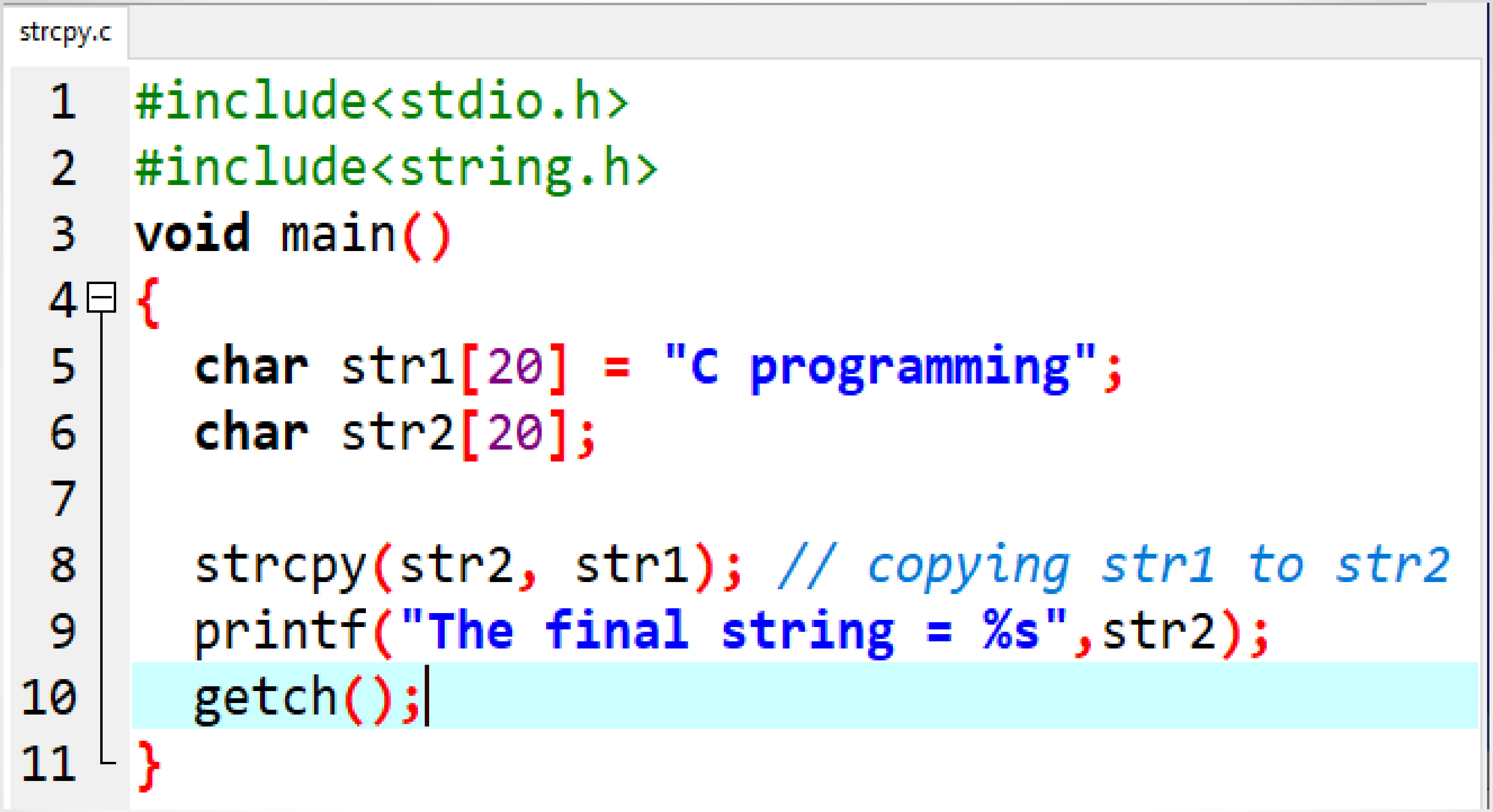


C:\Users\Admin\Desktop\Cpractical\strcm...

```
The value will be
0
-1
1
```

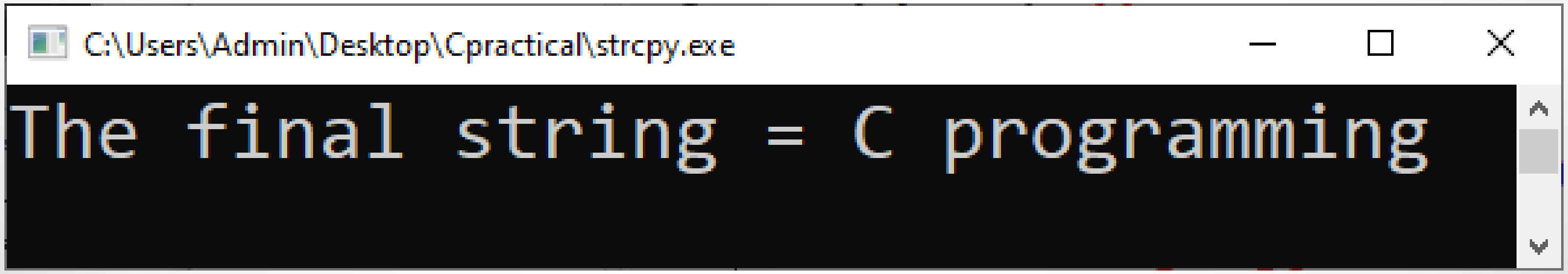
- **strcpy()**

- strcpy( ) function copies contents of one string into another string.
- Example:
  strcpy (str1, str2) – It copies contents of str2 into str1.
  strcpy (str2, str1) – It copies contents of str1 into str2.



strcpy.c

```c
#include<stdio.h>
#include<string.h>
void main()
{
   char str1[20] = "C programming";
   char str2[20];

   strcpy(str2, str1); // copying str1 to str2
   printf("The final string = %s",str2);
   getch();
}
```

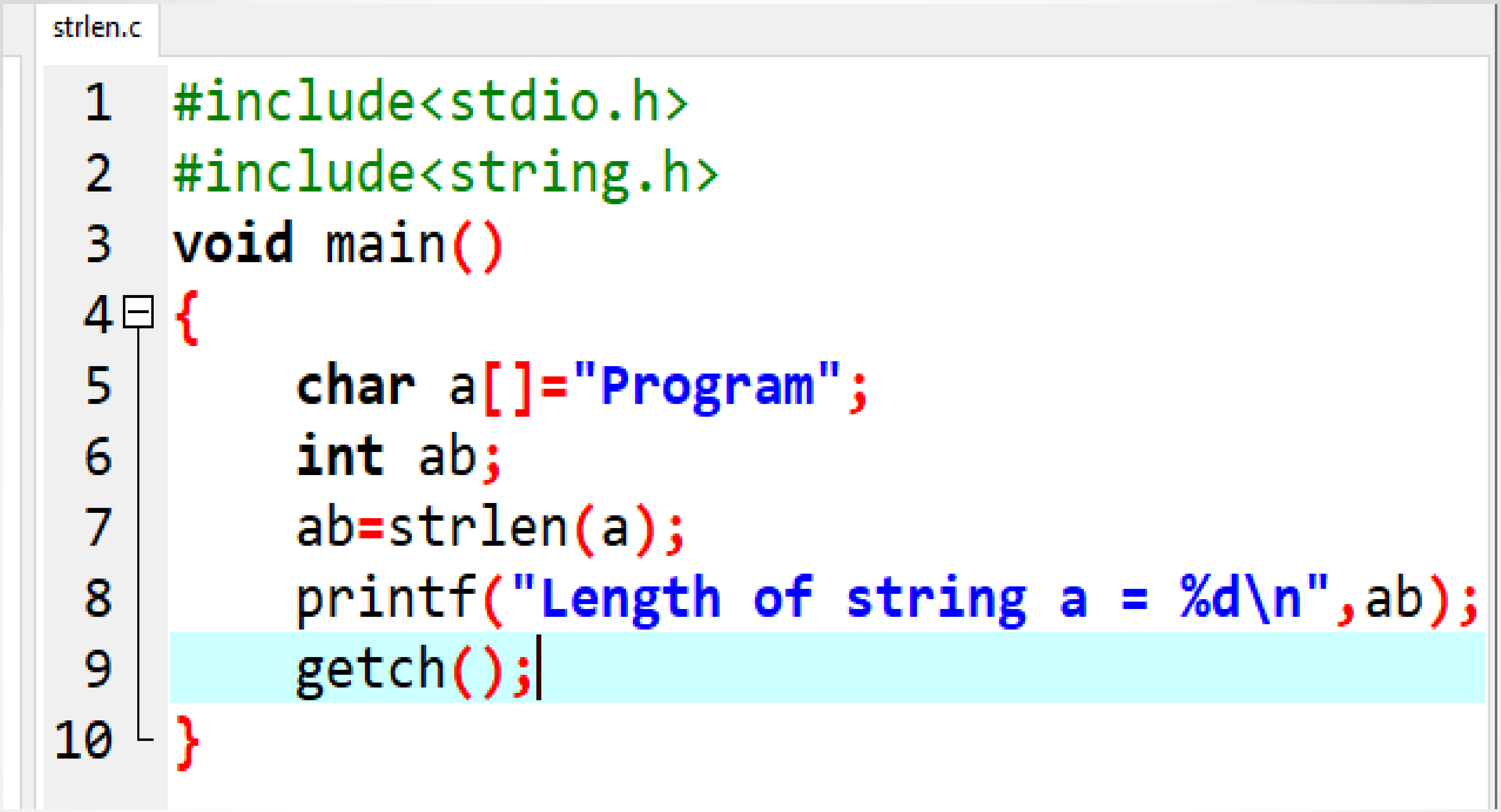


C:\Users\Admin\Desktop\Cpractical\strcpy.exe

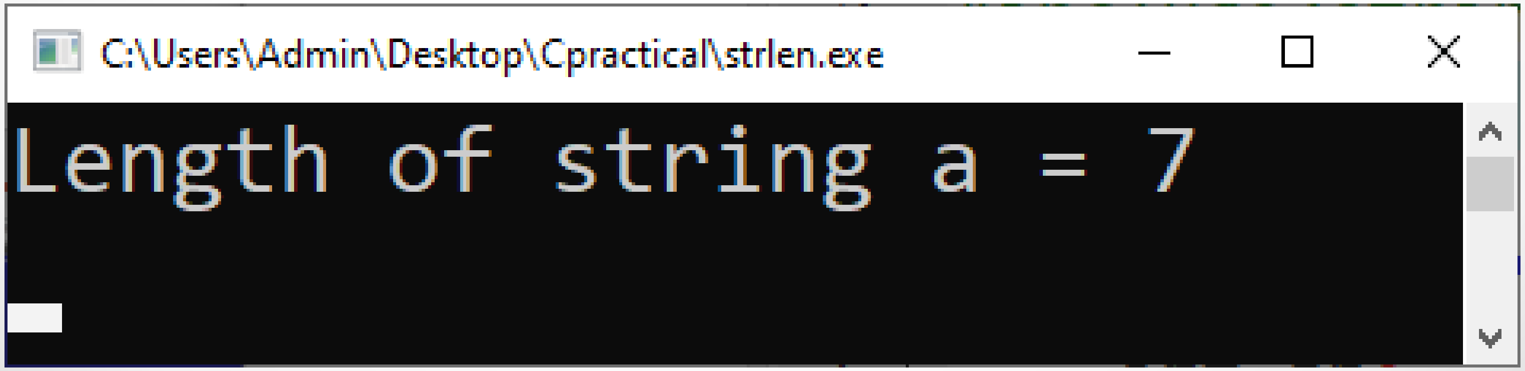
```
The final string = C programming
```

- **strlen()**

- strlen( ) function in C gives the length of the given string.
- strlen( ) function counts the number of characters in a given string and returns the integer value.

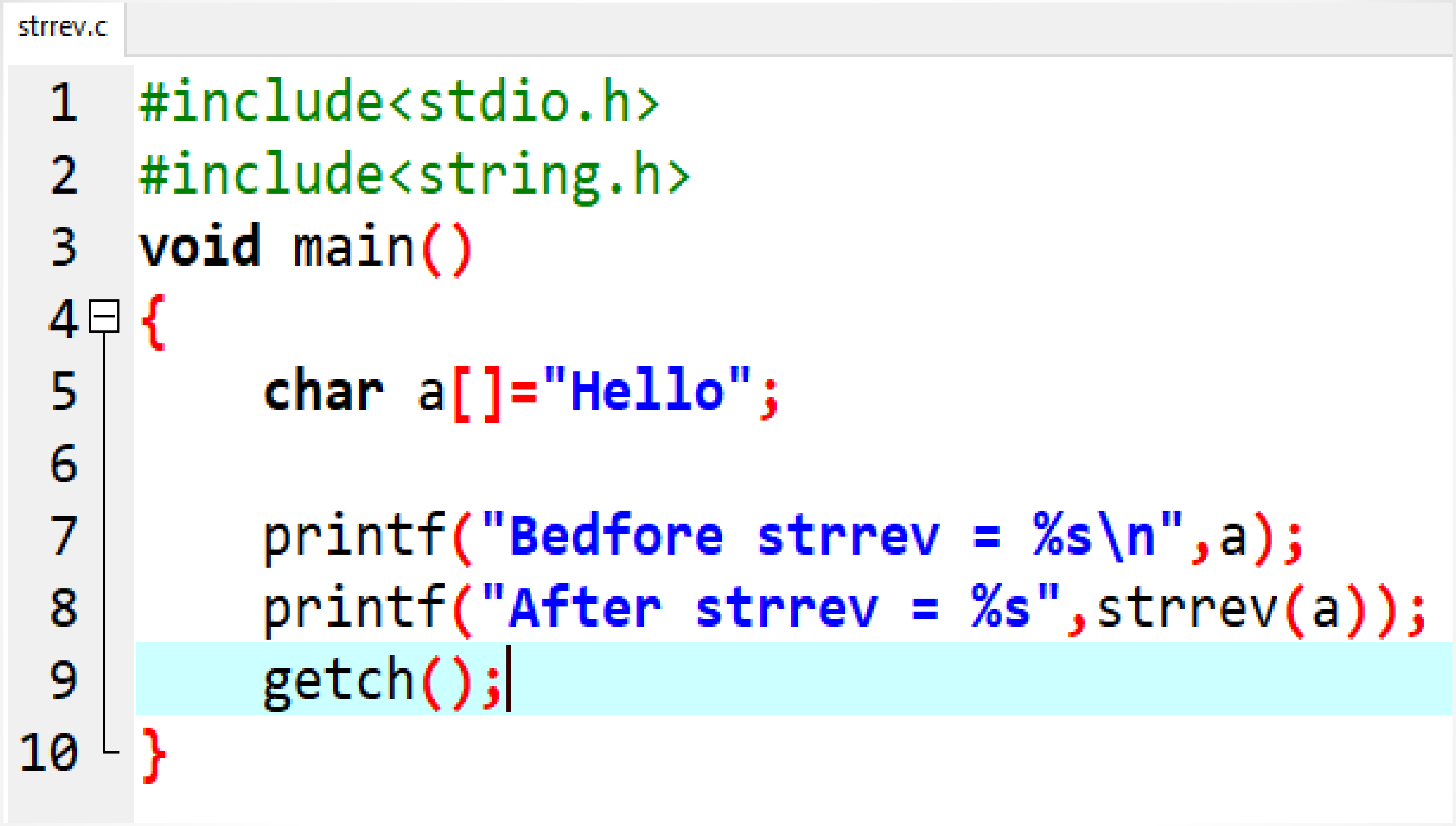

strlen.c

```c
#include<stdio.h>
#include<string.h>
void main()
{
	char a[]="Program";
	int ab;
	ab=strlen(a);
	printf("Length of string a = %d\n",ab);
	getch();
}
```


C:\Users\Admin\Desktop\Cpractical\strlen.exe

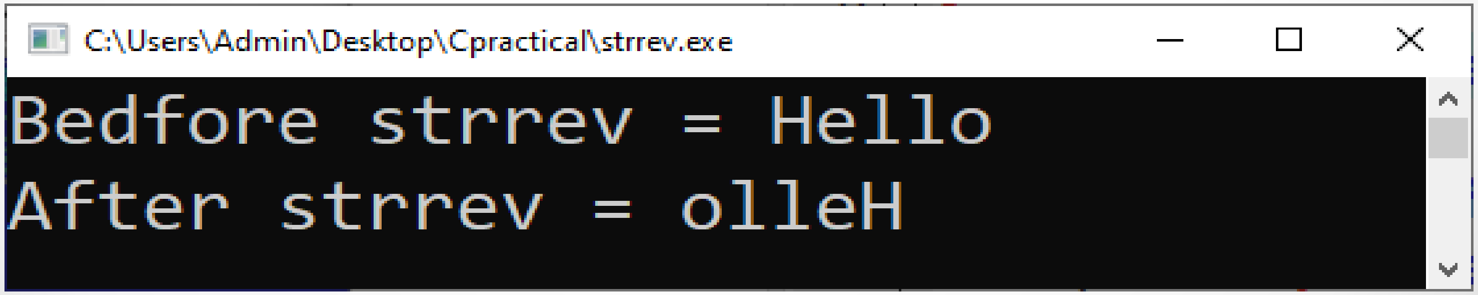
```
Length of string a = 7
```

➢ **strrev( )** - Reverses the given string



strrev.c

```c
#include<stdio.h>
#include<string.h>
void main()
{
    char a[]="Hello";

    printf("Bedfore strrev = %s\n",a);
    printf("After strrev = %s",strrev(a));
    getch();
}
```



C:\Users\Admin\Desktop\Cpractical\strrev.exe

```
Bedfore strrev = Hello
After strrev = olleH
```

(Mathematical Functions)

❖ **The header file required for mathematical function is <math.h>**

➢ **floor( )**

- This function returns the nearest integer which is less than or equal to the argument passed to this function.

➢ **ceil( )**

- This function returns nearest integer value which is greater than or equal to the argument passed to this function.

➢ **sqrt( )**

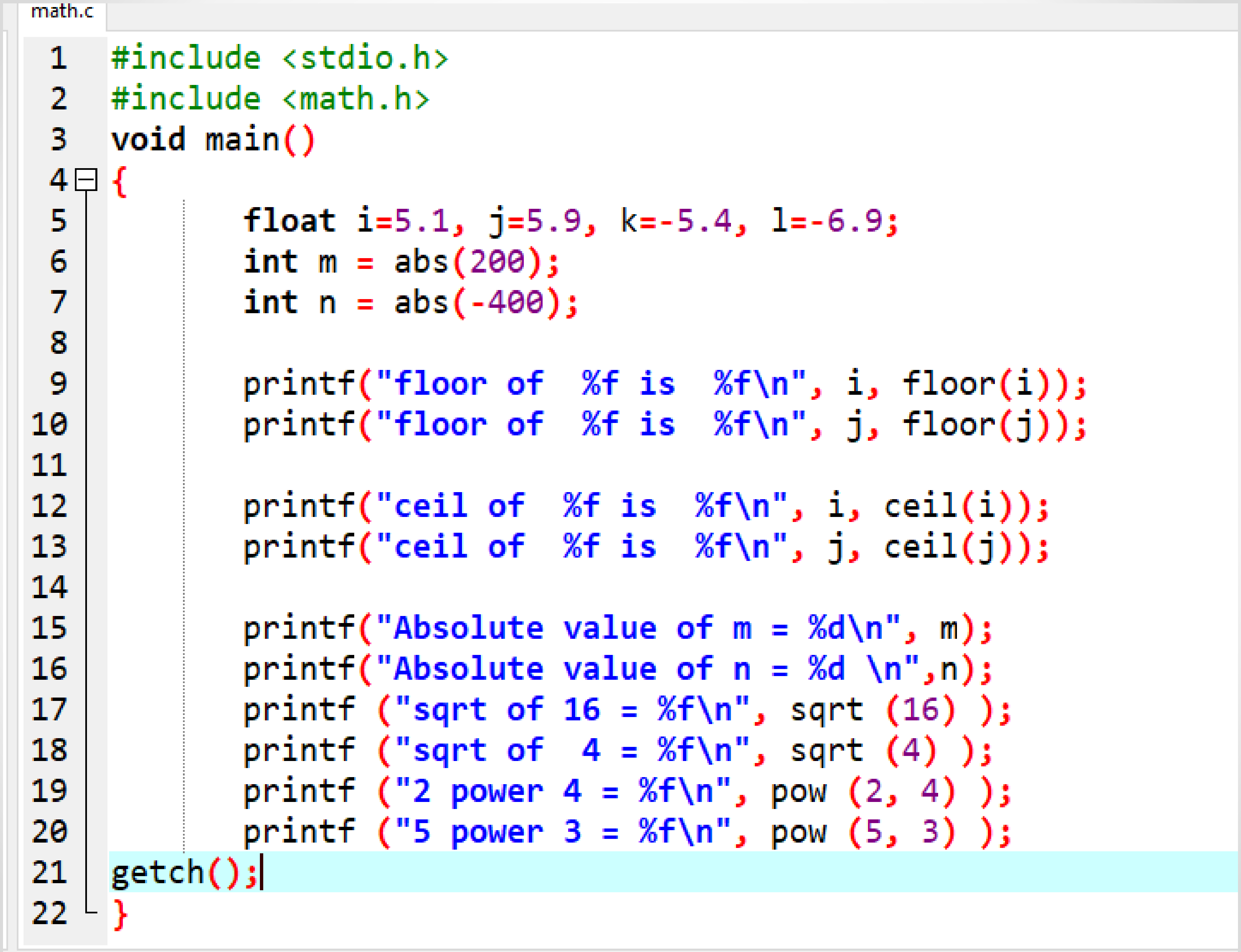
- This function is used to find square root of the argument passed to this function.

➢ **pow( )**

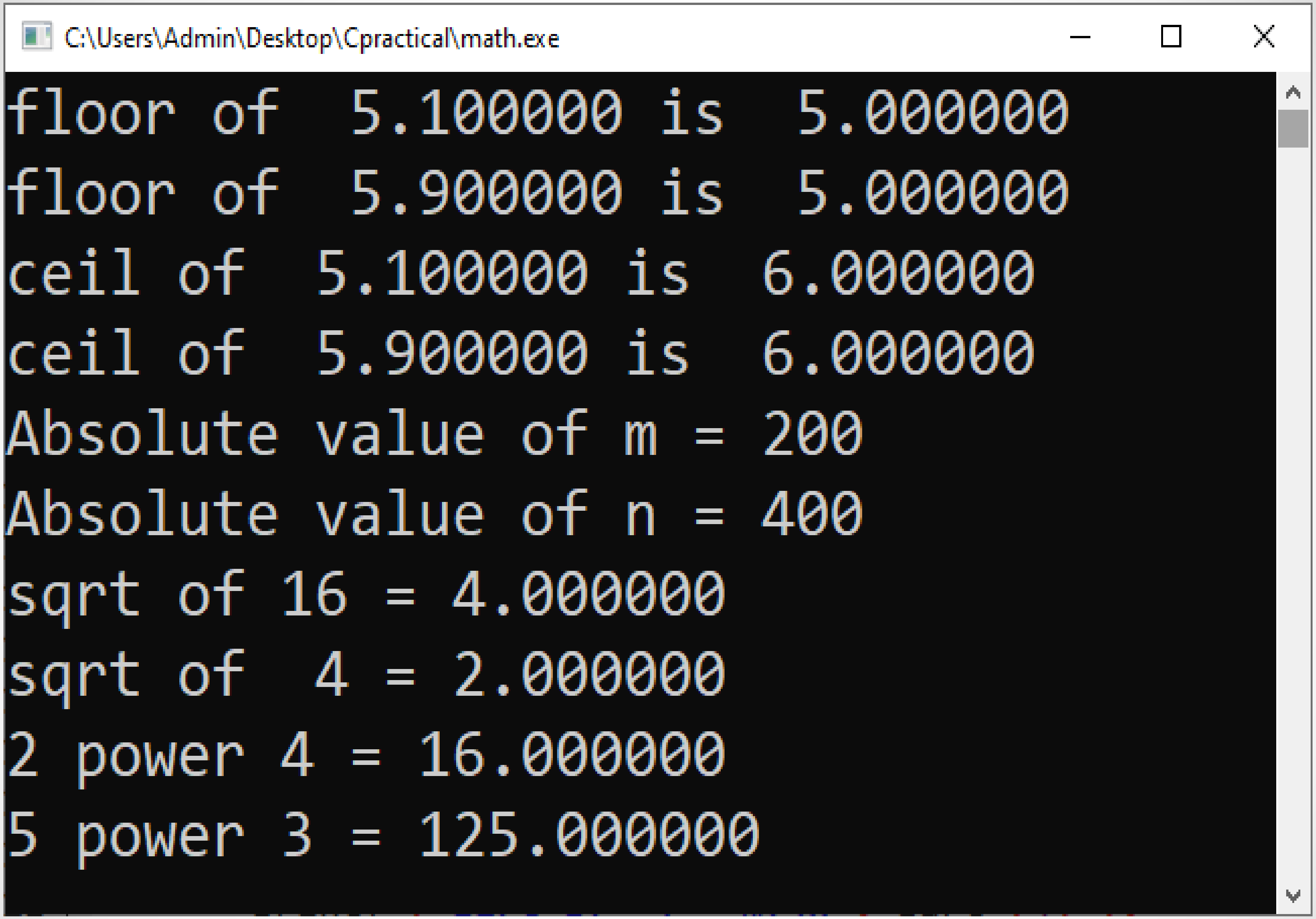
- This is used to find the power of the given number.

➢ **abs( )**

- Return the absolute value.


math.c

```c
#include <stdio.h>
#include <math.h>
void main()
{
        float i=5.1, j=5.9, k=-5.4, l=-6.9;
        int m = abs(200);
        int n = abs(-400);

        printf("floor of  %f is  %f\n", i, floor(i));
        printf("floor of  %f is  %f\n", j, floor(j));

        printf("ceil of  %f is  %f\n", i, ceil(i));
        printf("ceil of  %f is  %f\n", j, ceil(j));

        printf("Absolute value of m = %d\n", m);
        printf("Absolute value of n = %d \n",n);
        printf ("sqrt of 16 = %f\n", sqrt (16) );
        printf ("sqrt of  4 = %f\n", sqrt (4) );
        printf ("2 power 4 = %f\n", pow (2, 4) );
        printf ("5 power 3 = %f\n", pow (5, 3) );
getch();
}
```


C:\Users\Admin\Desktop\Cpractical\math.exe

```
floor of  5.100000 is  5.000000
floor of  5.900000 is  5.000000
ceil of  5.100000 is  6.000000
ceil of  5.900000 is  6.000000
Absolute value of m = 200
Absolute value of n = 400
sqrt of 16 = 4.000000
sqrt of  4 = 2.000000
2 power 4 = 16.000000
5 power 3 = 125.000000
```

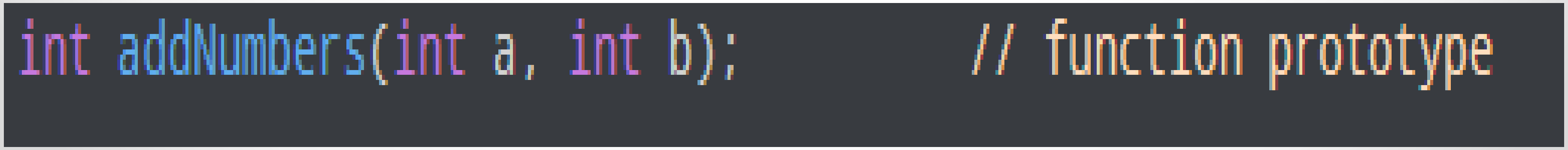
User-defined Function

Function Prototypes (*Function Declaration*)

- A function prototype is simply the declaration of a function that specifies ***function's name, parameters and return type***. It doesn't contain function body.
- This is also called as **function declaration**
- A function prototype gives information to the compiler that the function may later be used in the program
- Function prototypes are usually written at the ***beginning of a program***.
- The ***argument*** are also called as ***parameters***
- **Syntax**

return_type function_name(data_type argument list,…..);

- **Example**

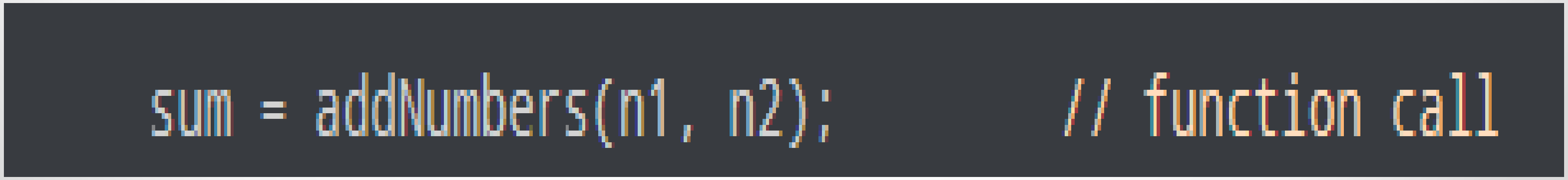


```
int addNumbers(int a, int b);          // function prototype
```

Accessing A Function (*Function Call*)

- A function can be accessed (i.e., called) by specifying its name, followed by a list of arguments enclosed in parentheses and separated by commas.
- This is also called **function call**
- It is **called inside a program whenever it is required to call a function**. It is **only called by its name** in the **main() function** of a program
- **Syntax**

  function_name(argument1, argument2, ...);
- **Example**



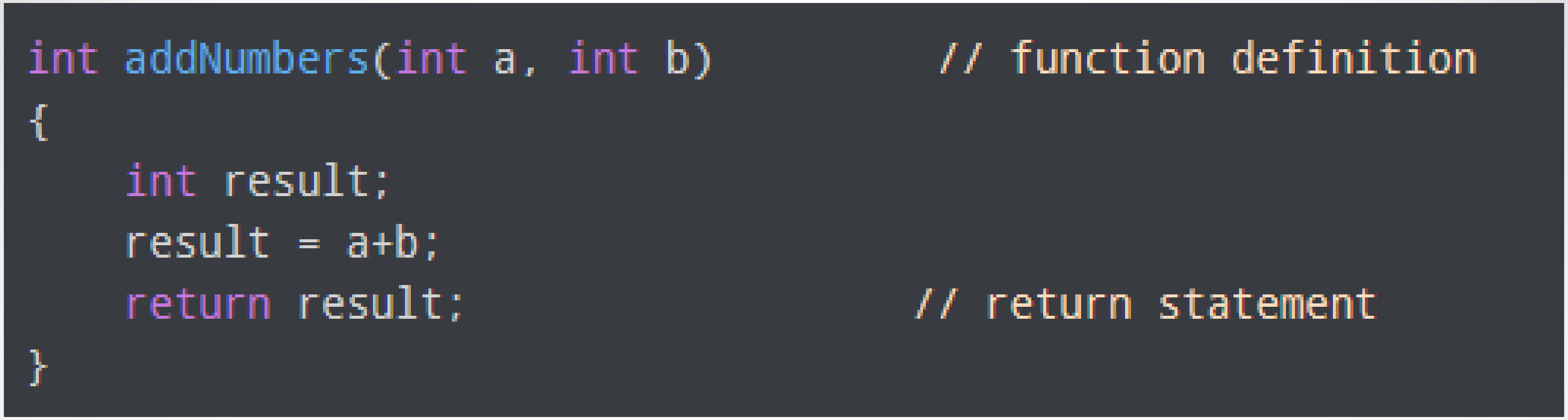
```
sum = addNumbers(n1, n2);          // function call
```

Defining Function (*Function Definition*)

- **Function definition** contains the **block of code** to perform **a specific task.**
- The actual task of the function is implemented in the **function definition**
- The function definition is also known as the **body of the function**
- **<u>Syntax</u>**

```
 return_type   function_name(data_type parameter,.....)
  {
        function body (i.e. code to be executed);
  }
```

- **Example**



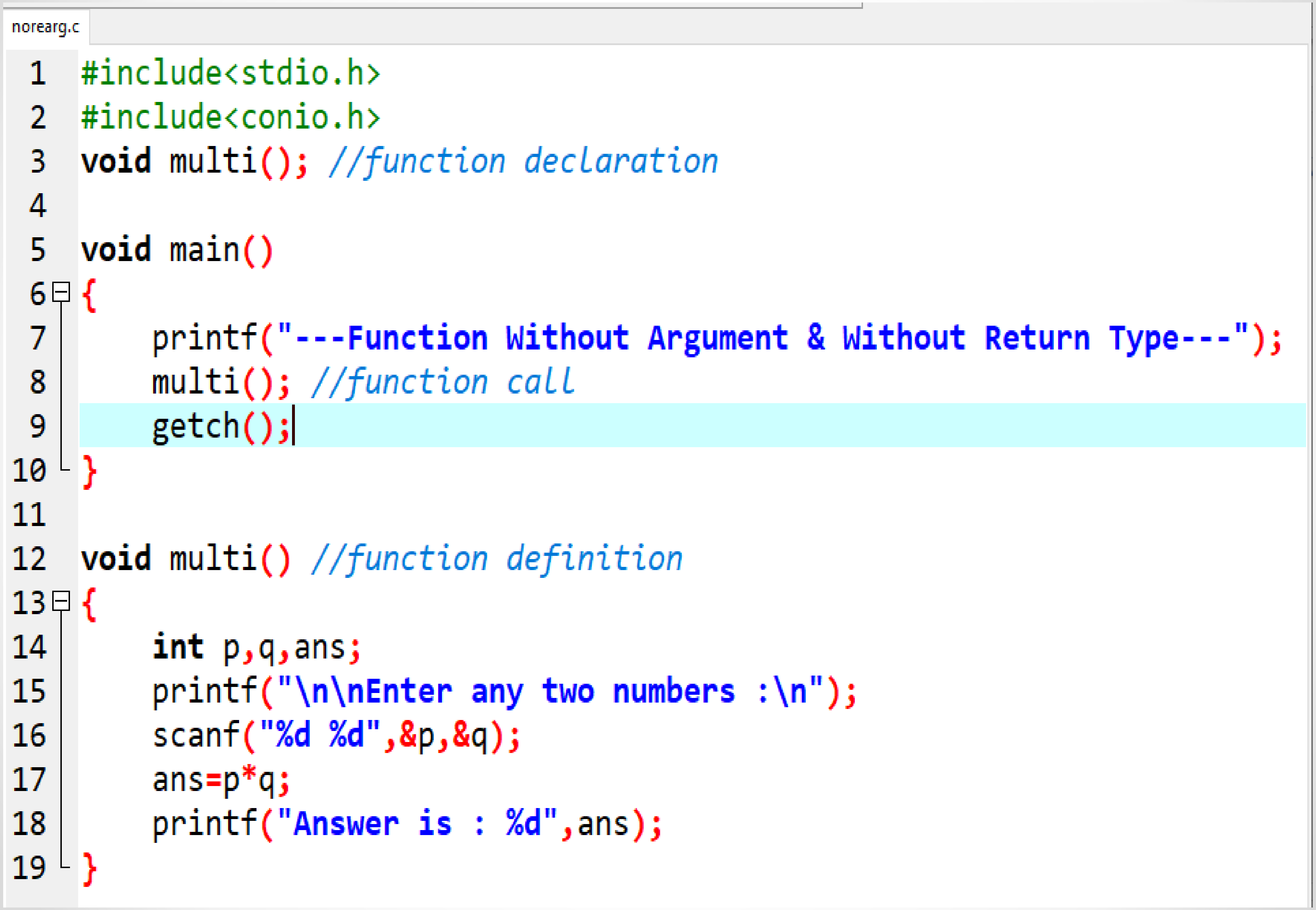
```
int addNumbers(int a, int b)         // function definition
{
    int result;
    result = a+b;
    return result;                  // return statement
}
```

- A **function definition** contains the parts as follow:
- **Return Data_Type:**
- It **defines the return data type of a value in the function**.
- The return data type can be integer, float, character, etc.
- Some functions perform the desired operations without returning a value. In this case, the return_type is the keyword **void**
- **Function Name**
- It defines the **actual name of a function** that contains some parameters.
- **Parameters/ Arguments**
- It is a parameter that **passed inside the function name** of a program.
- Parameters can be any type, order, and the number of parameters.
- **Function Body**
- It is the **collection of the statements to be executed** for performing the **specific tasks in a function**.

- **Function can be divided into 4 categories:**

1. Function without argument and without return value
2. Function without argument and with return value
3. Function with argument and without return value
4. Function with argument and with return value

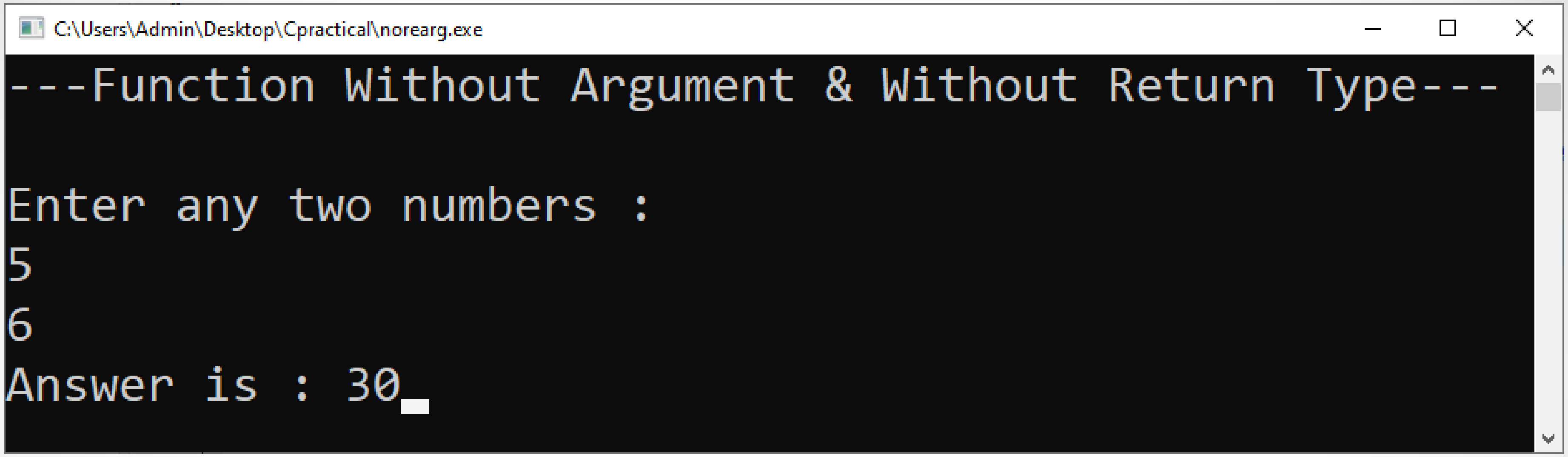
Function without argument and without return value



norearg.c

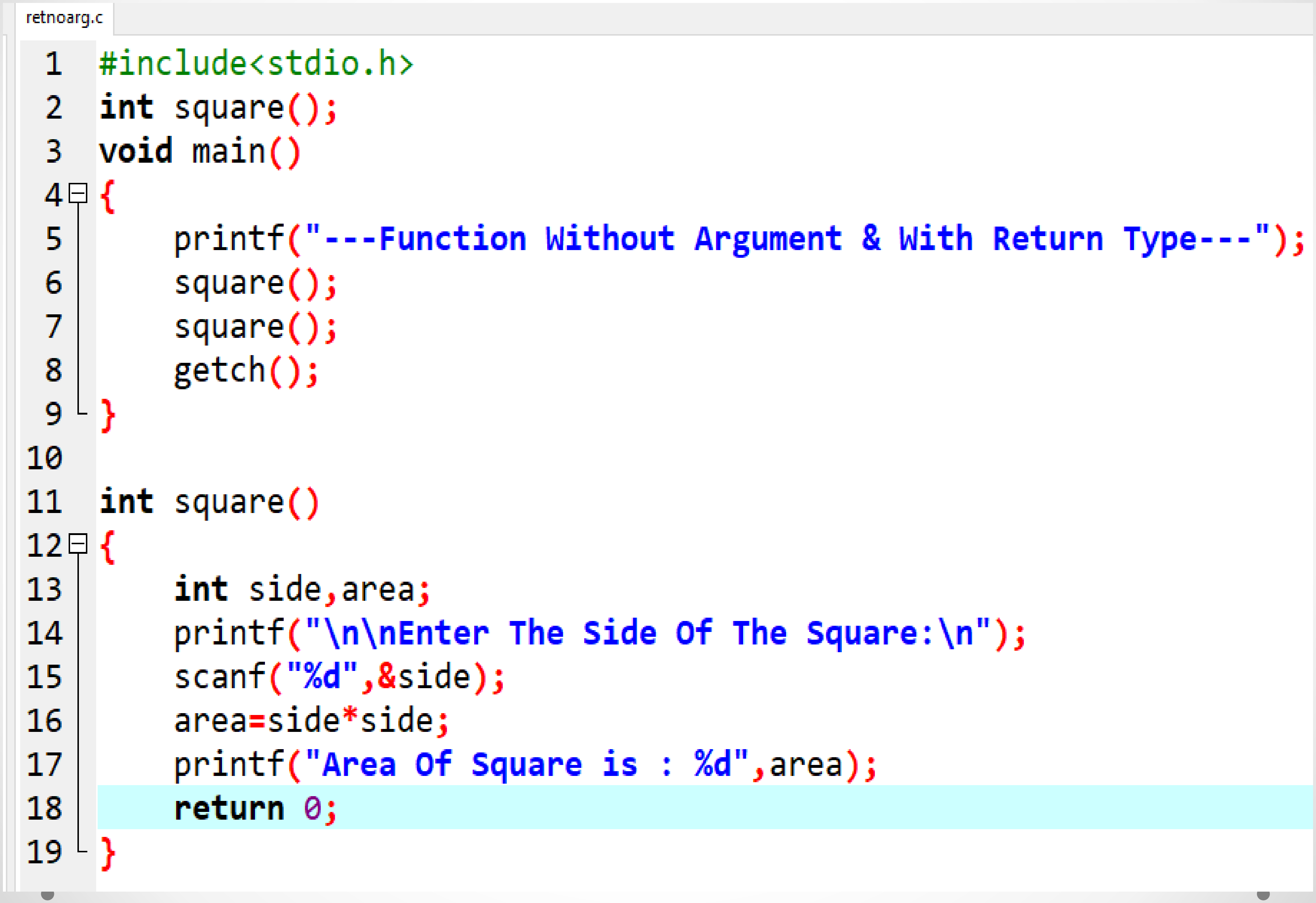
```
#include<stdio.h>
#include<conio.h>
void multi(); //function declaration

void main()
{
	printf("---Function Without Argument & Without Return Type---");
	multi(); //function call
	getch();
}

void multi() //function definition
{
	int p,q,ans;
	printf("\n\nEnter any two numbers :\n");
	scanf("%d %d",&p,&q);
	ans=p*q;
	printf("Answer is : %d",ans);
}
```

Output :


C:\Users\Admin\Desktop\Cpractical\norearg.exe

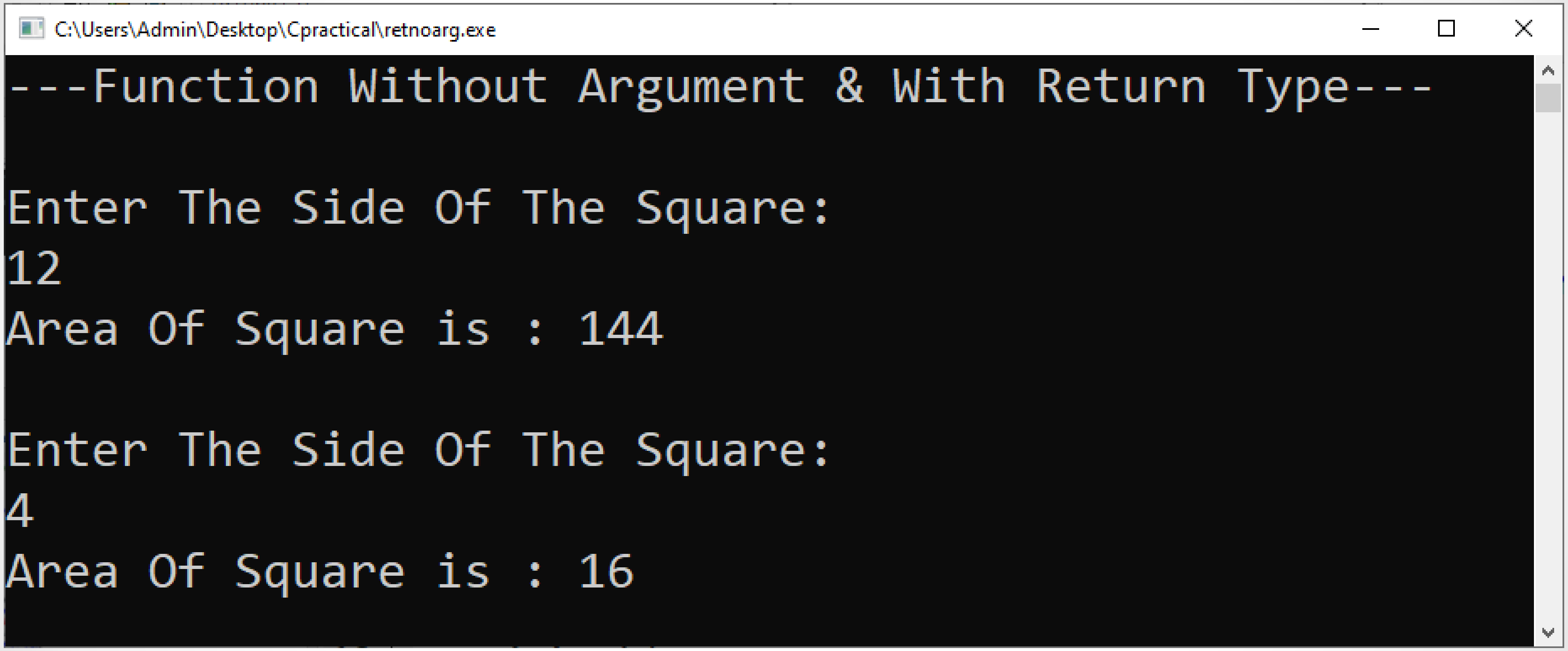
```
---Function Without Argument & Without Return Type---

Enter any two numbers :
5
6
Answer is : 30
```

Function without argument and with return value


retnoarg.c

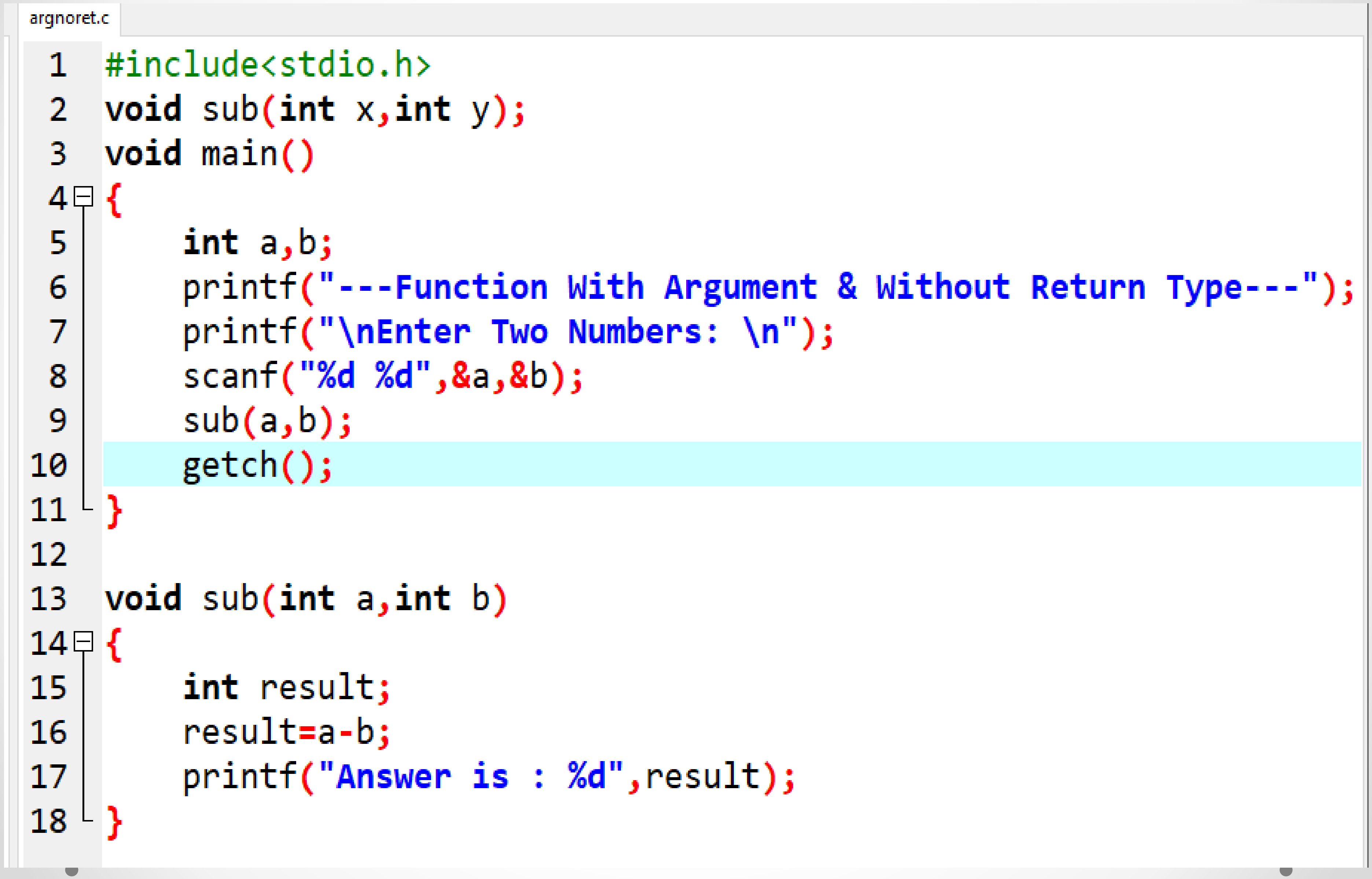
```c
#include<stdio.h>
int square();
void main()
{
    printf("---Function Without Argument & With Return Type---");
    square();
    square();
    getch();
}

int square()
{
    int side,area;
    printf("\n\nEnter The Side Of The Square:\n");
    scanf("%d",&side);
    area=side*side;
    printf("Area Of Square is : %d",area);
    return 0;
}
```

Output :



```
---Function Without Argument & With Return Type---

Enter The Side Of The Square:
12
Area Of Square is : 144

Enter The Side Of The Square:
4
Area Of Square is : 16
```

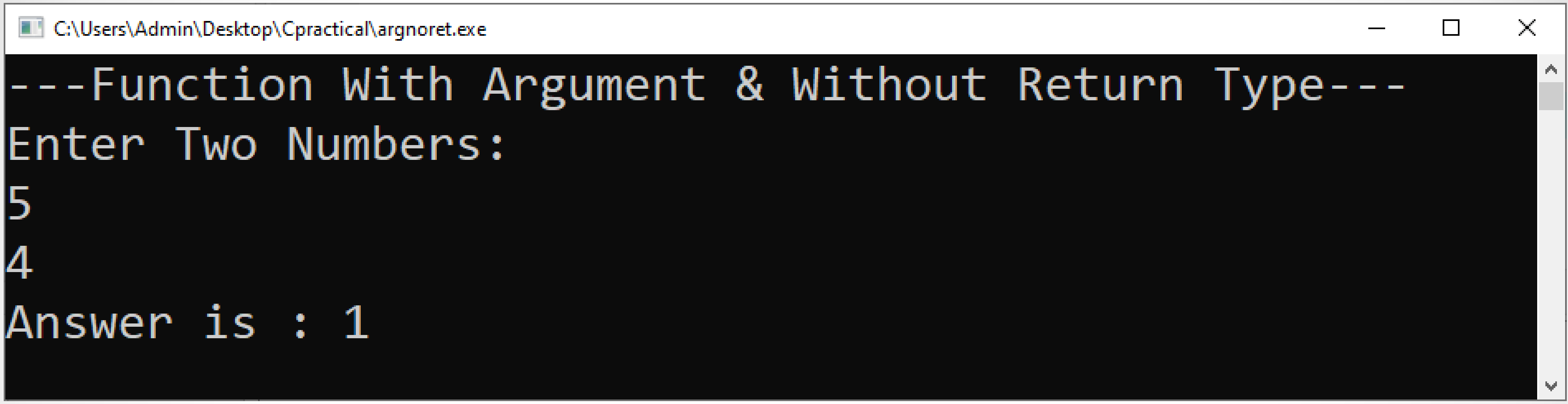
Function with argument and without return value



argnoret.c

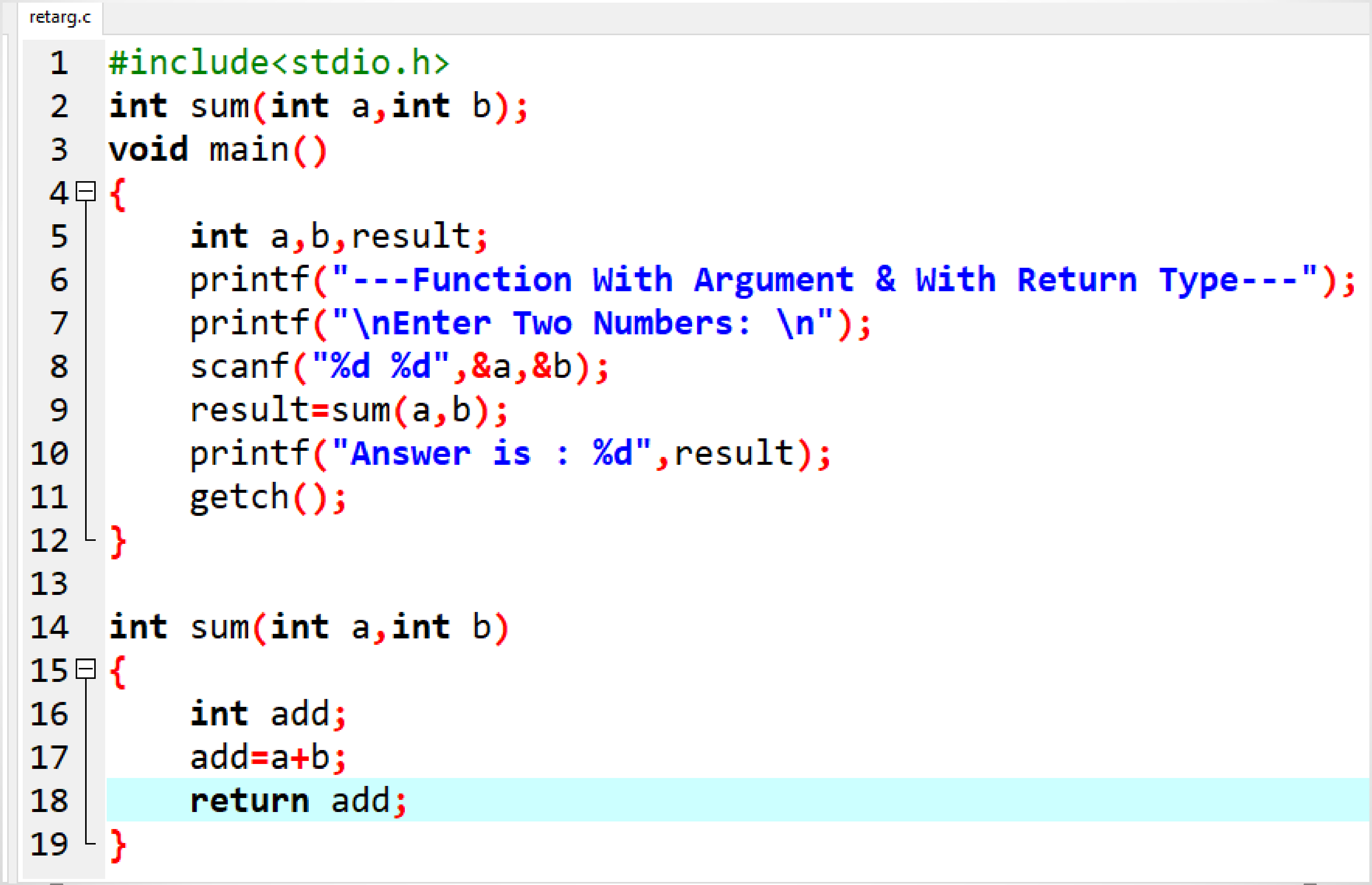
```c
#include<stdio.h>
void sub(int x,int y);
void main()
{
    int a,b;
    printf("---Function With Argument & Without Return Type---");
    printf("\nEnter Two Numbers: \n");
    scanf("%d %d",&a,&b);
    sub(a,b);
    getch();
}

void sub(int a,int b)
{
    int result;
    result=a-b;
    printf("Answer is : %d",result);
}
```

Output :



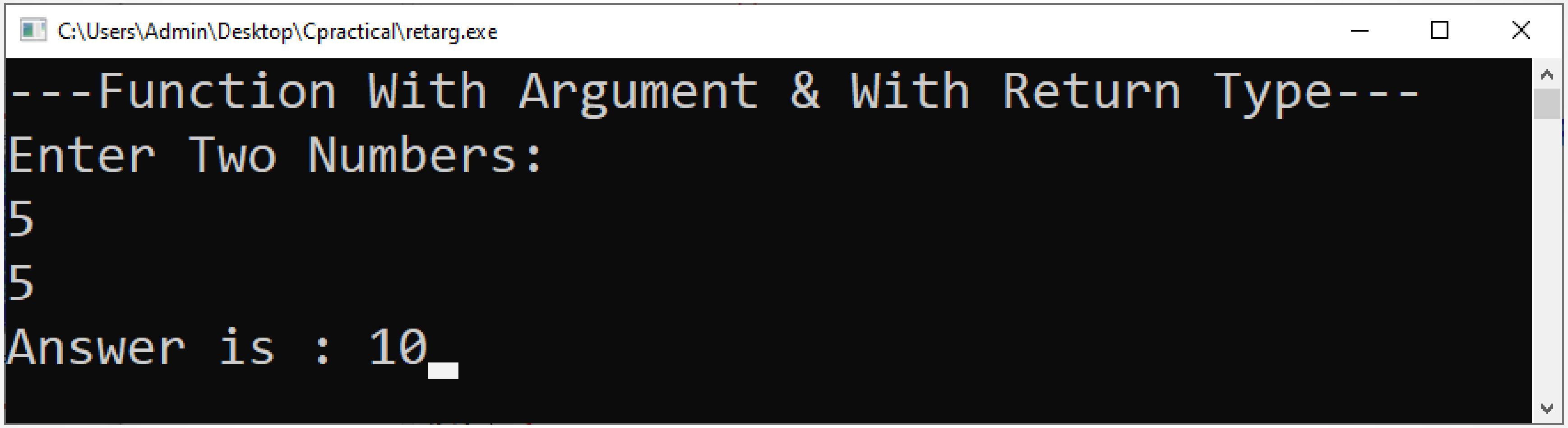
```
---Function With Argument & Without Return Type---
Enter Two Numbers:
5
4
Answer is : 1
```

Function with argument and with return value



```c
#include<stdio.h>
int sum(int a,int b);
void main()
{
	int a,b,result;
	printf("---Function With Argument & With Return Type---");
	printf("\nEnter Two Numbers: \n");
	scanf("%d %d",&a,&b);
	result=sum(a,b);
	printf("Answer is : %d",result);
	getch();
}

int sum(int a,int b)
{
	int add;
	add=a+b;
	return add;
}
```

Output :


C:\Users\Admin\Desktop\Cpractical\retarg.exe

```
---Function With Argument & With Return Type---
Enter Two Numbers:
5
5
Answer is : 10
```

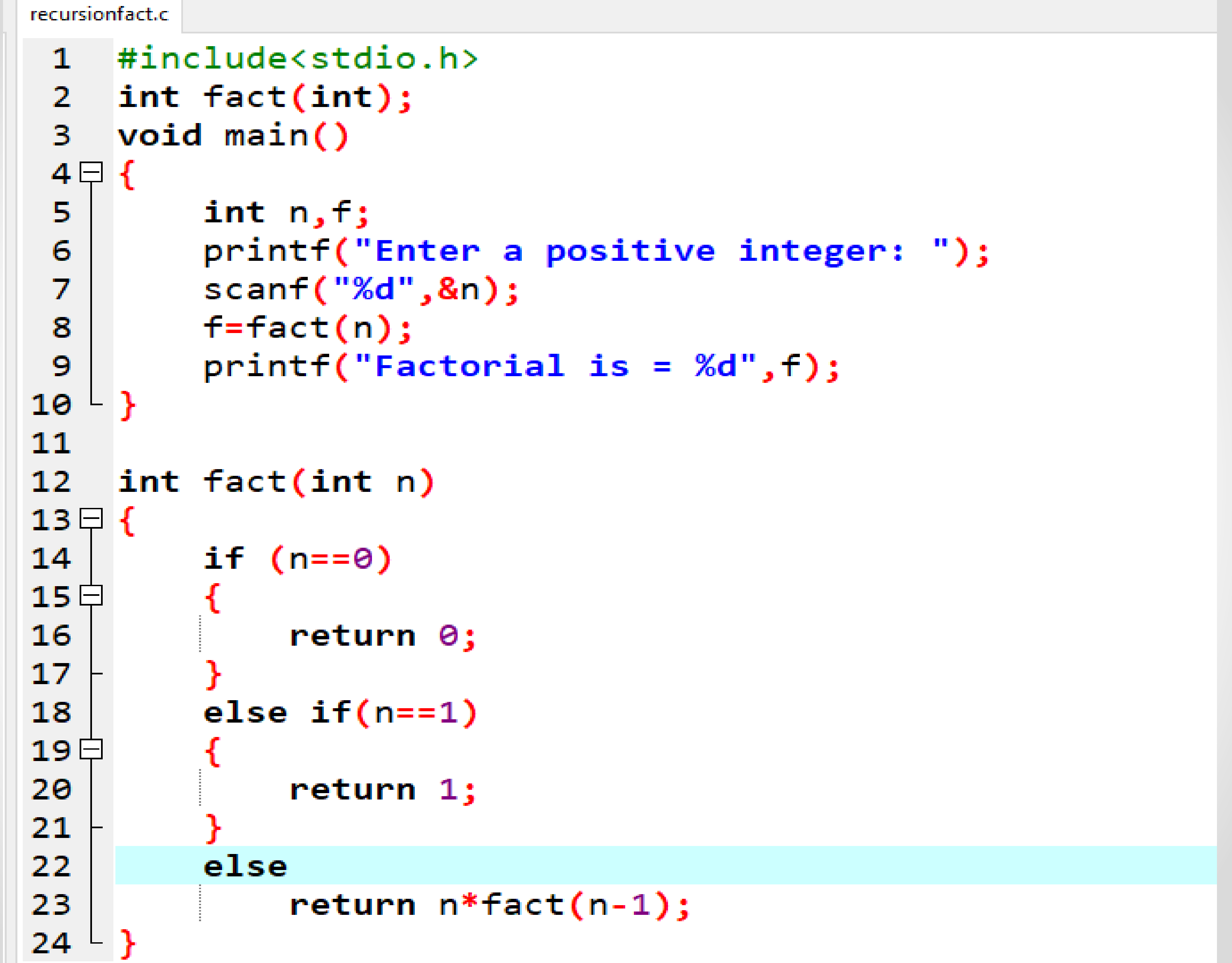
Recursion

- In C Programming, if a function calls itself from inside, the same function is called recursion.
- Recursion is a process by which a function calls itself repeatedly
- The function which calls itself is called a recursive function, and the function call is termed a recursive call
- Recursion involves several numbers of recursive calls
- The recursion is similar to iteration but more complex.

- ❖ What are the differences between recursion and iteration

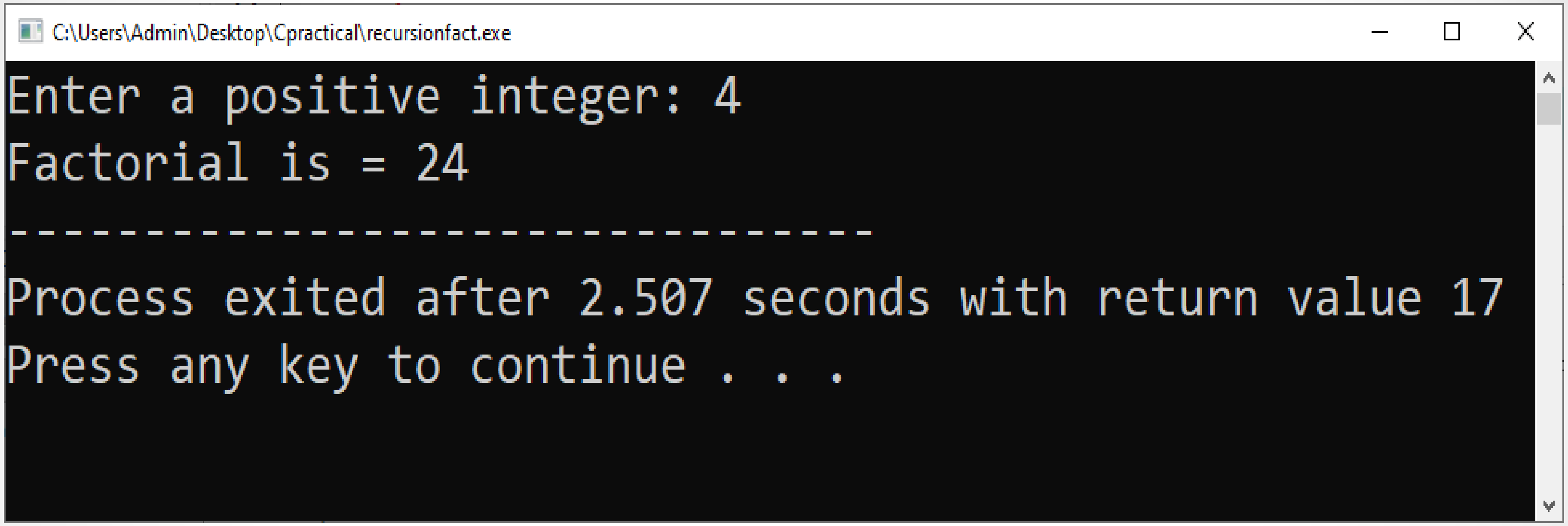
- ❑ **Recursion**
- Recursion is usually **slower than iteration** due to the overhead of maintaining the stack.
- Recursion uses **more memory than iteration**.
- Recursion makes the **code smaller**.

- ❑ **Iteration**
- An iteration **terminates** when the **loop condition fails**.
- An iteration does not use the **stack** so it's **faster than recursion**.
- Iteration consumes **less memory.**
- Iteration makes the **code longer**.


recursionfact.c

```
#include<stdio.h>
int fact(int);
void main()
{
	int n,f;
	printf("Enter a positive integer: ");
	scanf("%d",&n);
	f=fact(n);
	printf("Factorial is = %d",f);
}

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

Output :


C:\Users\Admin\Desktop\Cpractical\recursionfact.exe

```
Enter a positive integer: 4
Factorial is = 24
--------------------------------
Process exited after 2.507 seconds with return value 17
Press any key to continue . . .
```

Storage Class

- Storage class specifiers in C language tells the compiler **where to store a variable, how to store the variable, what is the initial value of the variable and life time of the variable.**
- **Syntax:**

storage_specifier data_type variable _name;

- There are four storage class specifiers available in C language. They are

✓ auto

✓ extern

✓ static

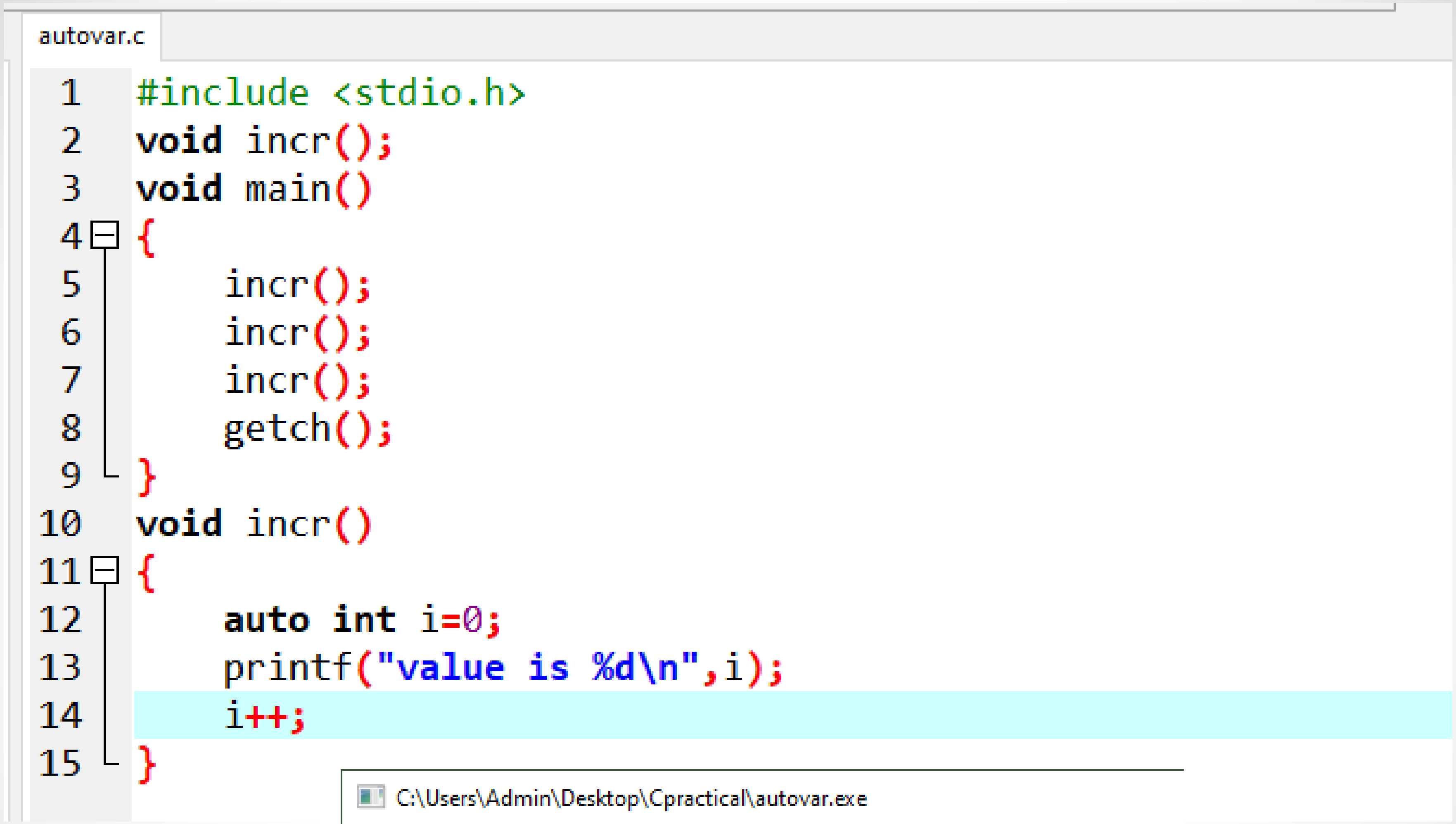
✓ register

Storage Classes	Storage Place	Default Value	Scope	Lifetime
auto	RAM	Garbage Value	Local	Within function
extern	RAM	Zero	Global	Till the end of the main program Maybe declared anywhere in the program
static	RAM	Zero	Local	Retains value between multiple functions call
register	Register	Garbage Value	Local	Within the function

Auto Variables

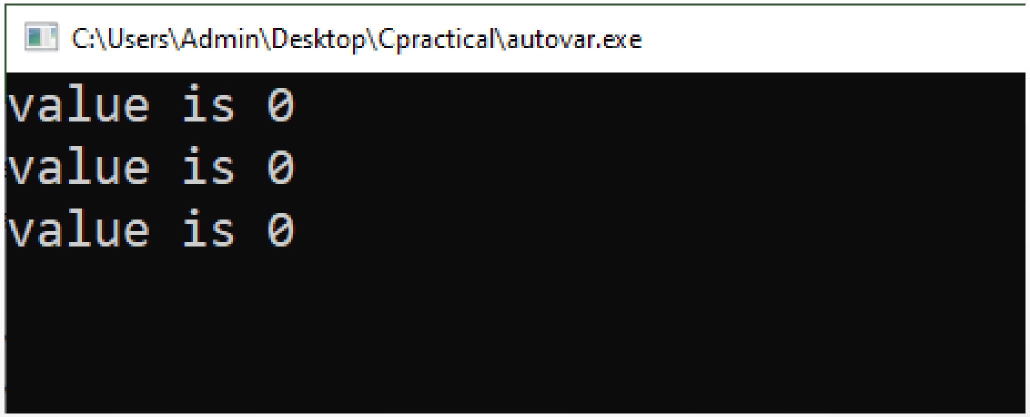
- The scope of this auto variable is within the function only.
- It is equivalent to local variable. All local variables are auto variables by default.

e.g.



autovar.c

```c
#include <stdio.h>
void incr();
void main()
{
     incr();
     incr();
     incr();
     getch();
}
void incr()
{
     auto int i=0;
     printf("value is %d\n",i);
     i++;
}
```



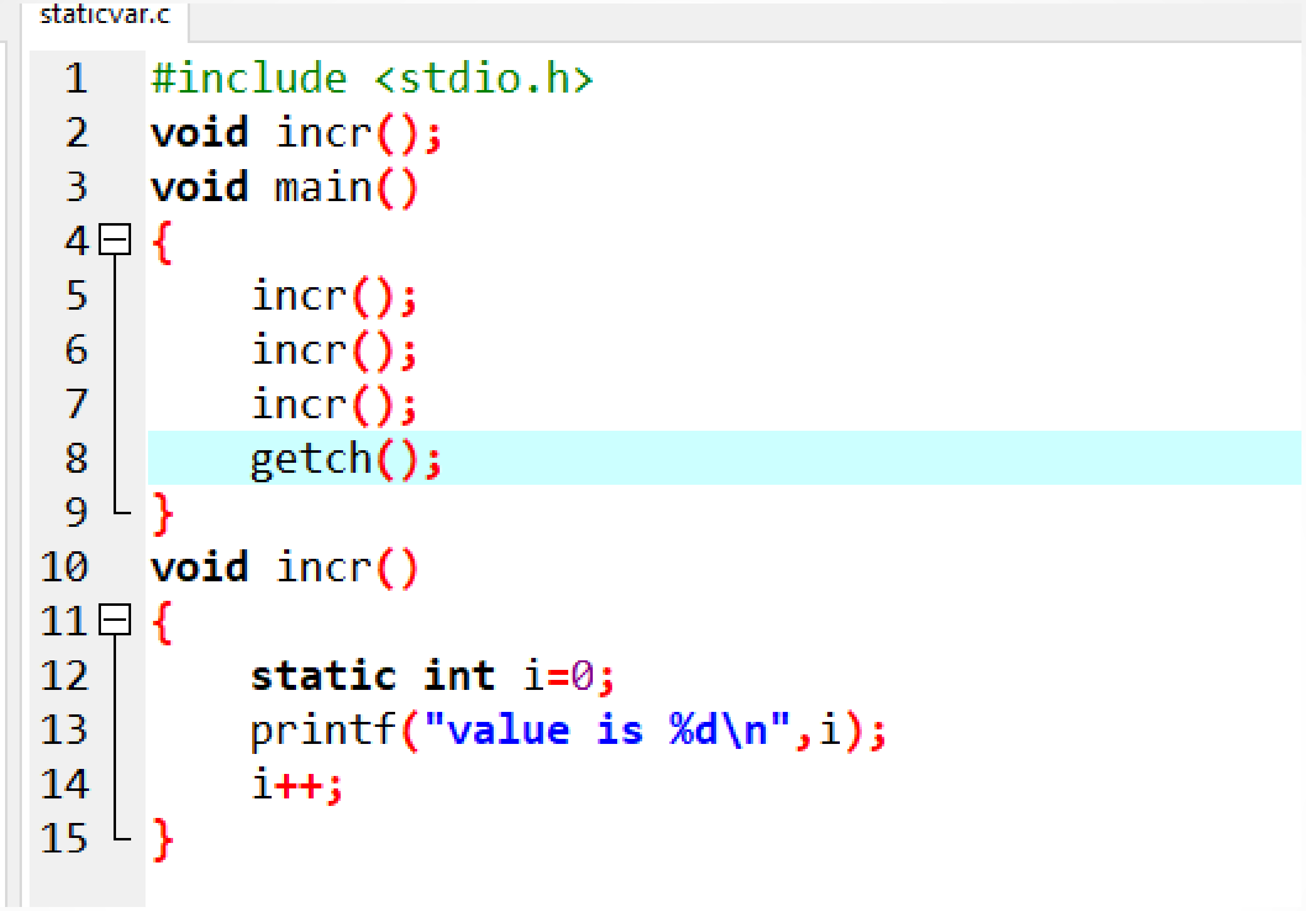
C:\Users\Admin\Desktop\Cpractical\autovar.exe

```
value is 0
value is 0
value is 0
```

Static Variables

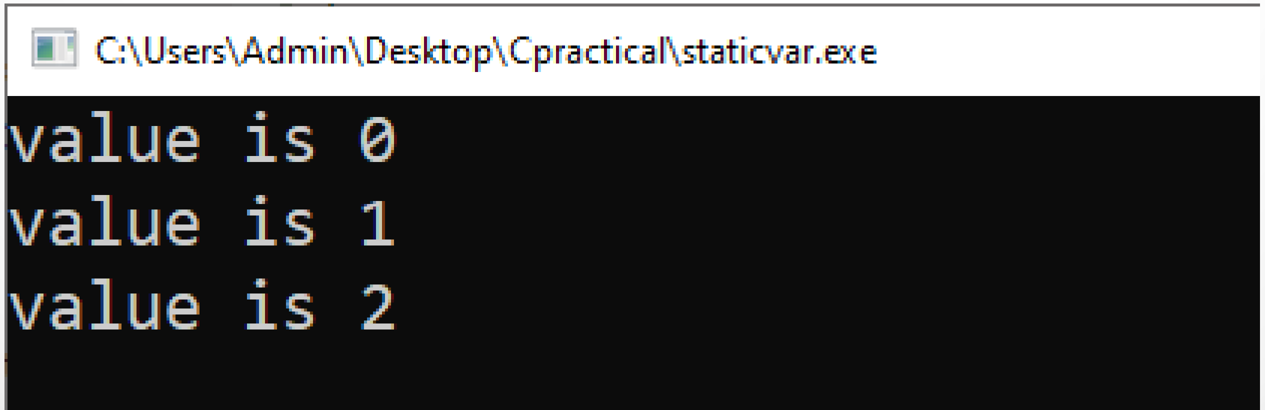
- Static variables retain the value of the variable between different function calls.

e.g.



staticvar.c

```c
#include <stdio.h>
void incr();
void main()
{
     incr();
     incr();
     incr();
     getch();
}
void incr()
{
     static int i=0;
     printf("value is %d\n",i);
     i++;
}
```



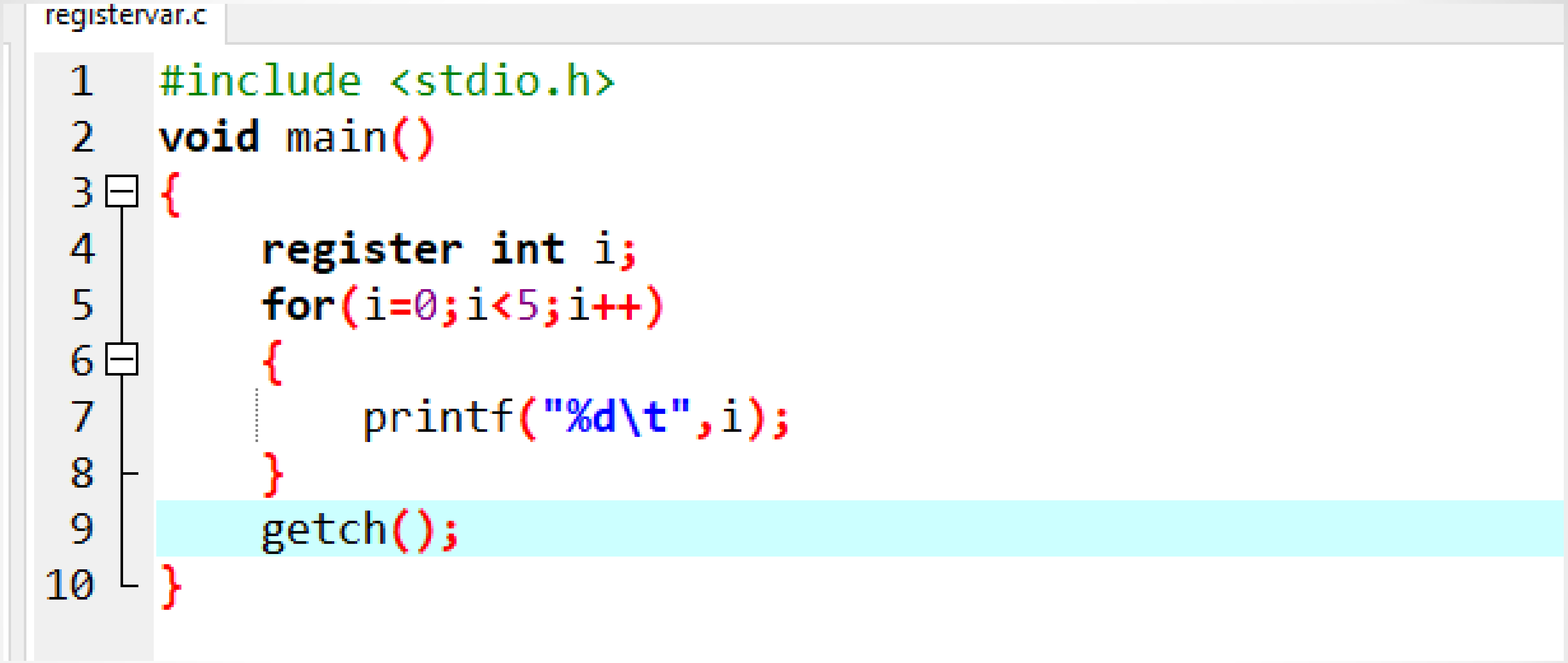
C:\Users\Admin\Desktop\Cpractical\staticvar.exe

```
value is 0
value is 1
value is 2
```

Register Variable

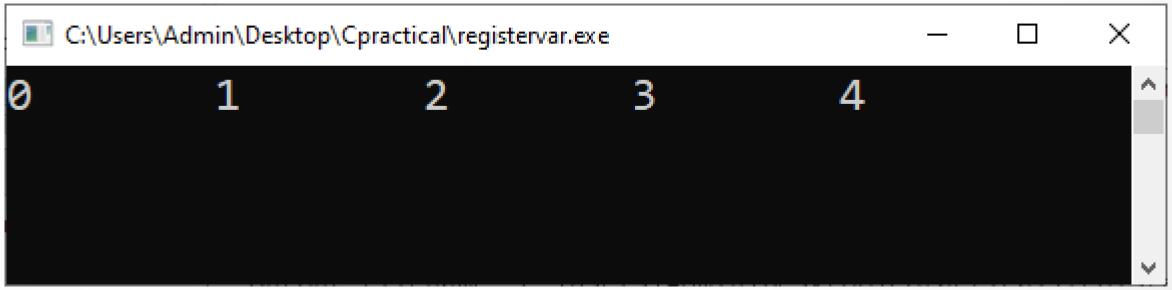
- Register variables are also local variables, but stored in register memory. Whereas, auto variables are stored in main CPU memory.
- Register variables will be accessed very faster than the normal variables since they are stored in register memory rather than main memory.

e.g.



registervar.c

```c
#include <stdio.h>
void main()
{
    register int i;
    for(i=0;i<5;i++)
    {
        printf("%d\t",i);
    }
    getch();
}
```



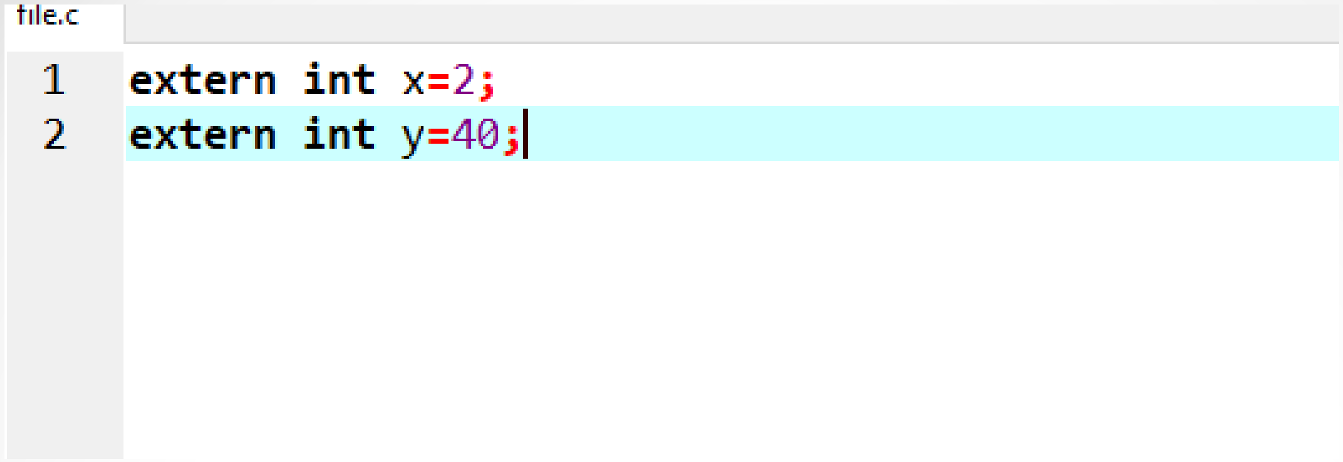
C:\Users\Admin\Desktop\Cpractical\registervar.exe

```
0       1       2       3       4
```

Extern Variables

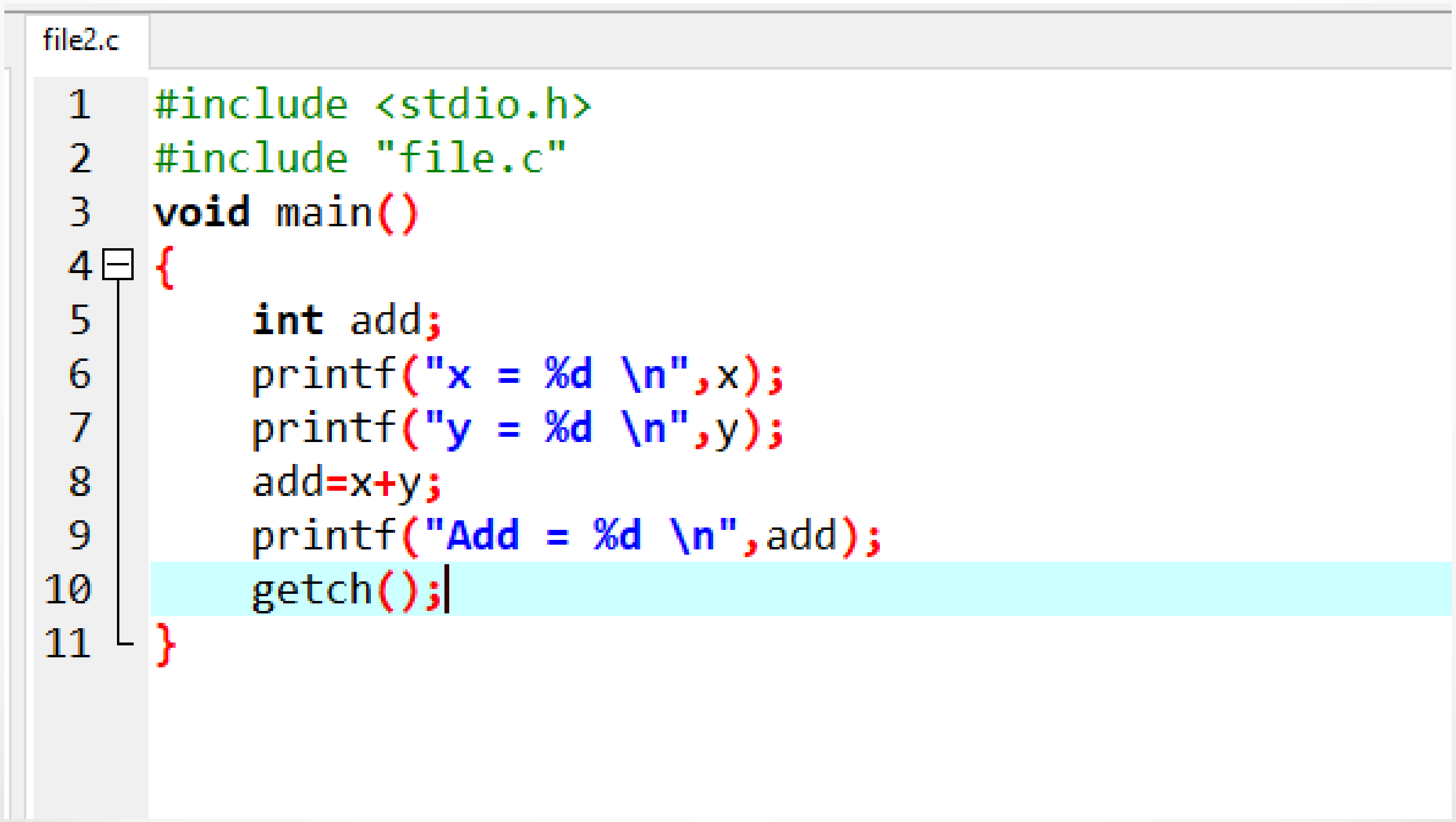
- The scope of this extern variable is throughout the main program. It is equivalent to global variable.
- Definition for extern variable might be anywhere in the C program.

e.g.



file.c

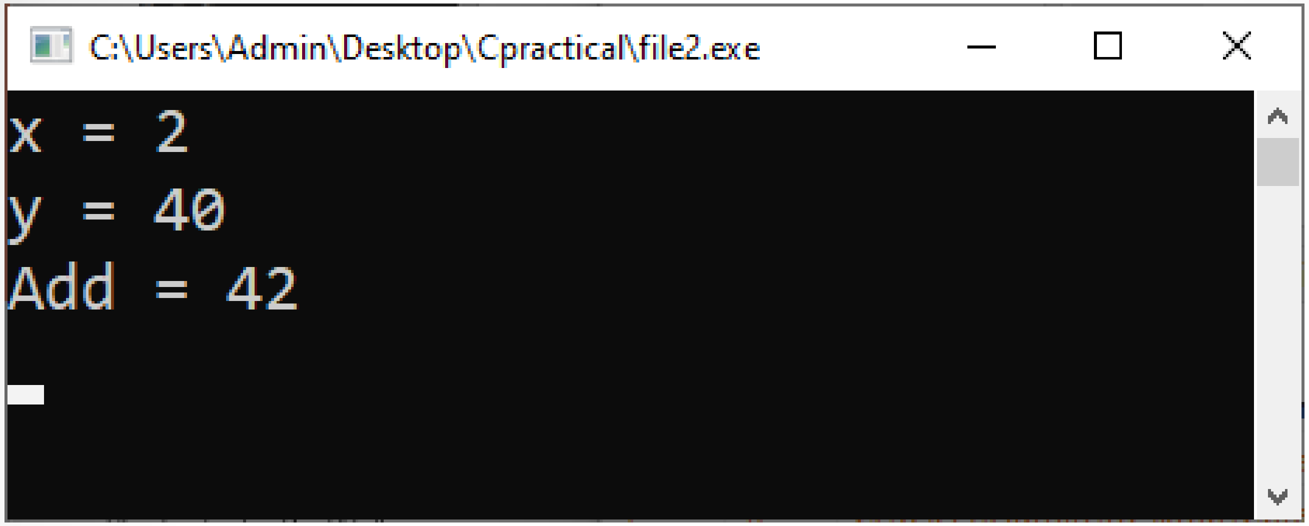
```
extern int x=2;
extern int y=40;
```



file2.c

```
#include <stdio.h>
#include "file.c"
void main()
{
    int add;
    printf("x = %d \n",x);
    printf("y = %d \n",y);
    add=x+y;
    printf("Add = %d \n",add);
    getch();
}
```

Output :



```
x = 2
y = 40
Add = 42
```

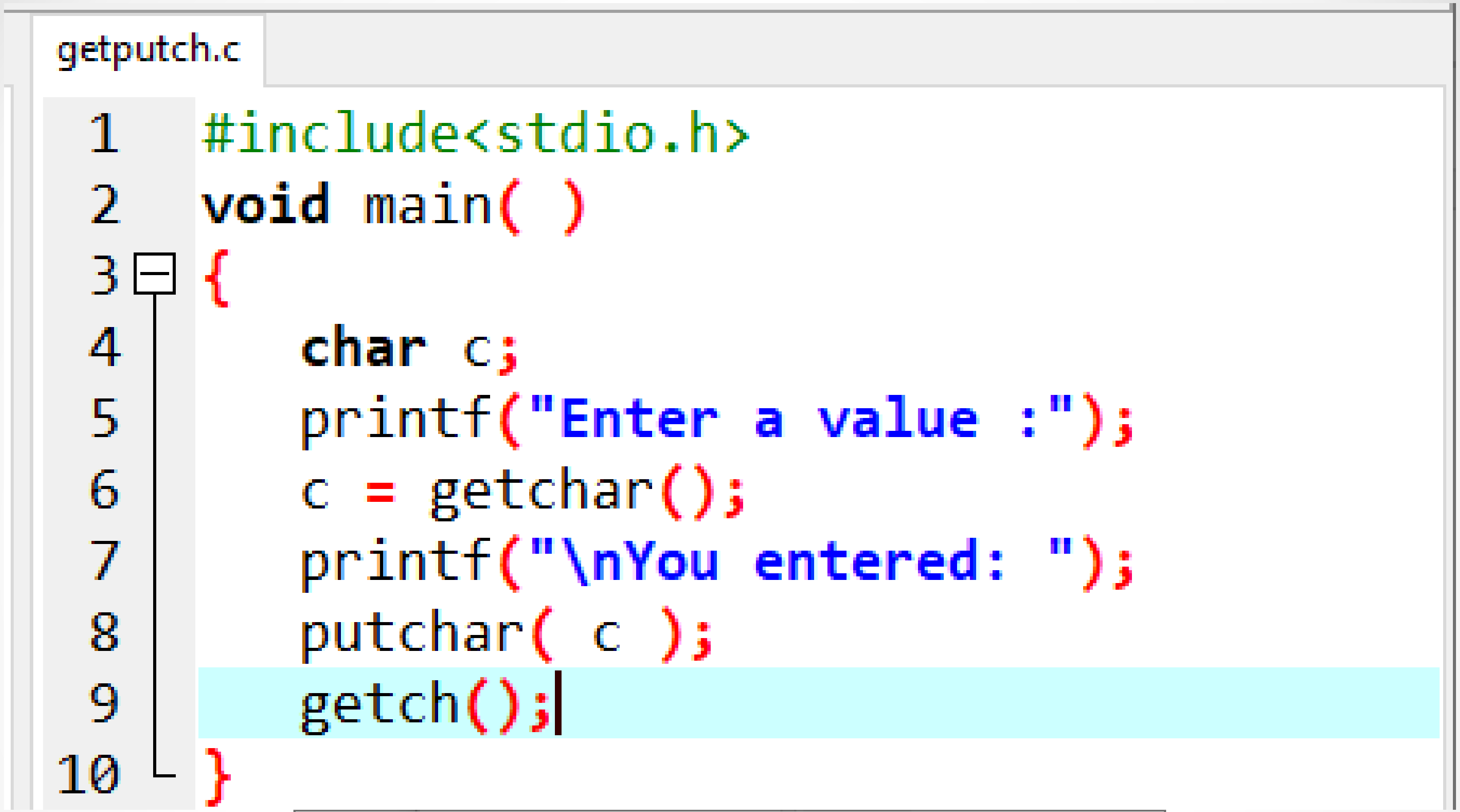
Standard Input/Output

- **Input**, it means to feed some data into a program
- **Output**, it means to display some data on screen
- C programming provides a set of **built-in functions to read the given input and feed it to the program as per requirement**
- C programming provides a set of **built-in functions to output the data on the computer screen.**
- C language is accompanied by a collection of library functions, which includes a number of input/output functions.
- In this we will make use of six of these functions: getchar, putchar, scanf, printf, gets and puts.
- These six functions permit the transfer of information between the computer and the standard input/output devices

- The ***getchar*** function
- **Single characters can be entered** into the computer using the C library function getchar
- It returns a single character from a standard input device (typically a keyboard)

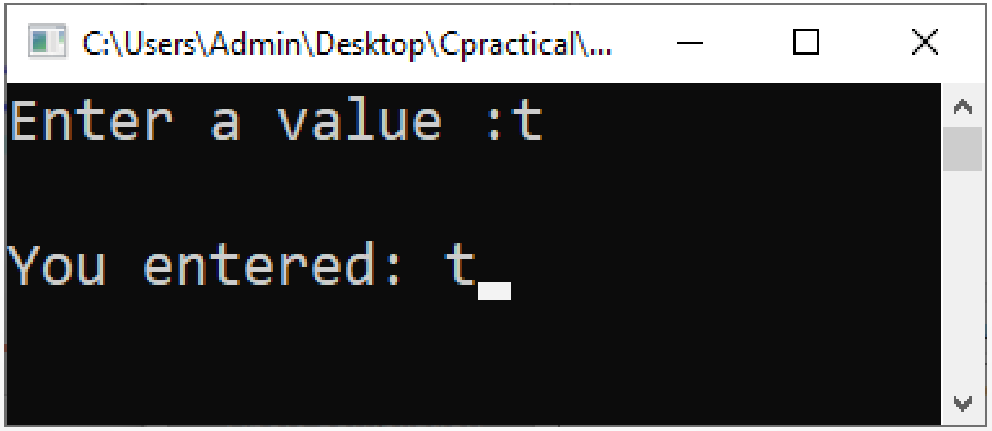
- The ***putchar*** function
- **Single characters can be displayed** (i.e. written out of the computer) using the C library function putchar
- It transmits a single character to a standard output device (typically computer monitor)

e.g.



getputch.c

```c
#include<stdio.h>
void main( )
{
    char c;
    printf("Enter a value :");
    c = getchar();
    printf("\nYou entered: ");
    putchar( c );
    getch();
}
```



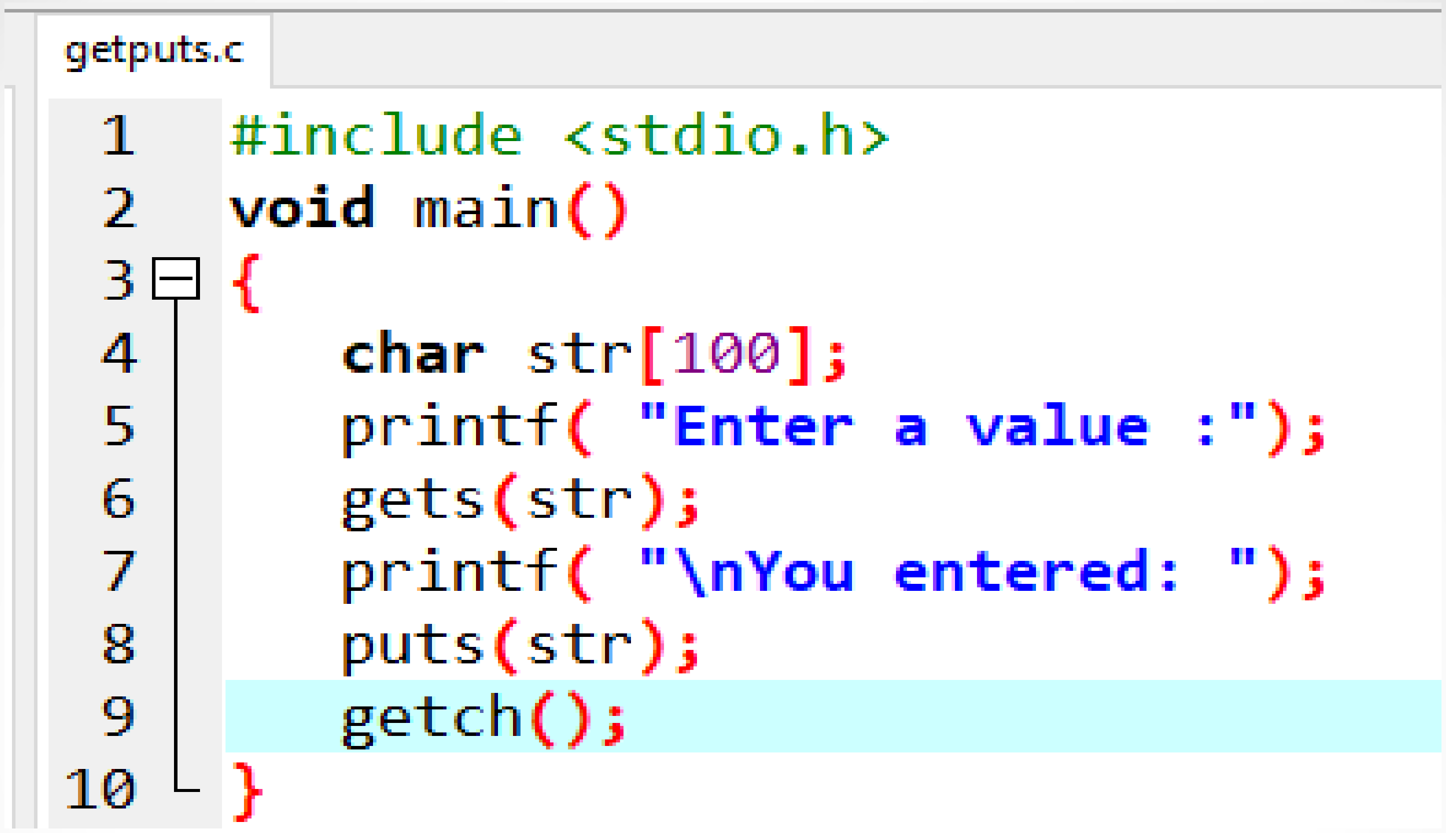
C:\Users\Admin\Desktop\Cpractical\...

```
Enter a value :t

You entered: t
```

❖ **The gets and puts function**

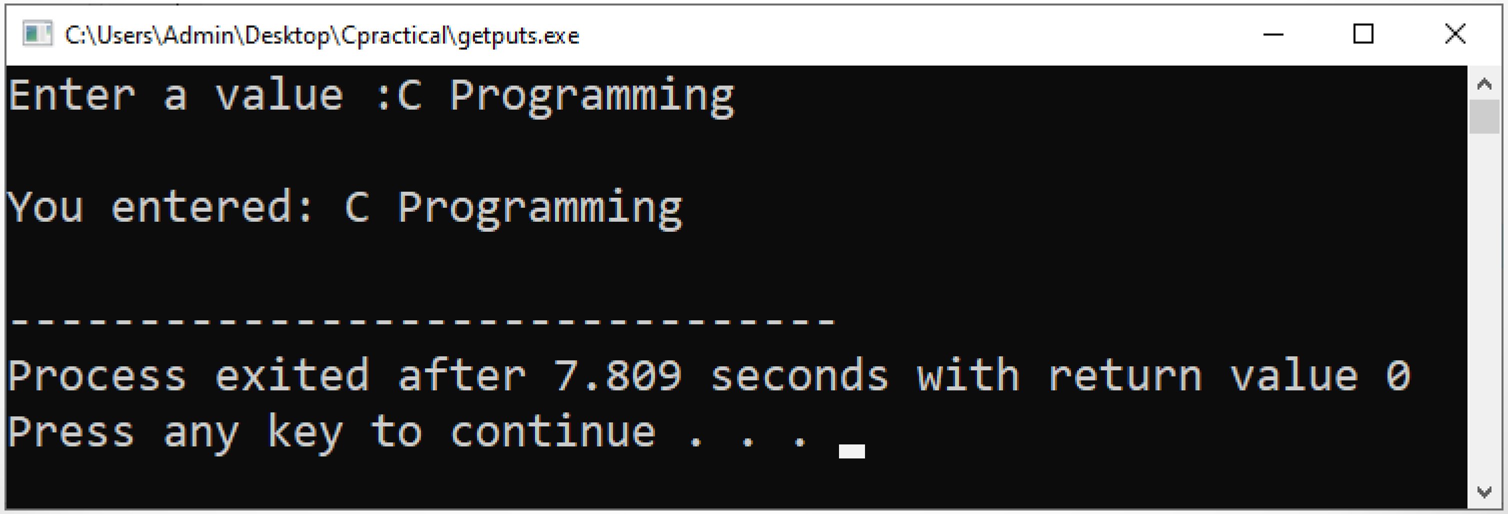
- The *gets* and *puts* functions, which facilitate the transfer of **strings** between the computer and the standard input/output devices.
- Each of these functions accepts a single argument.
- The argument must be a data item that represents a string
- The **gets() function can read** a full **string even blank spaces presents in a string.**
- The gets() function is used to **get any string from the user.**
- The **puts() function prints** the **character array or string on the console.**

e.g.



getputs.c

```c
#include <stdio.h>
void main()
{
    char str[100];
    printf( "Enter a value :");
    gets(str);
    printf( "\nYou entered: ");
    puts(str);
    getch();
}
```



C:\Users\Admin\Desktop\Cpractical\getputs.exe

```
Enter a value :C Programming

You entered: C Programming


--------------------------------
Process exited after 7.809 seconds with return value 0
Press any key to continue . . .
```

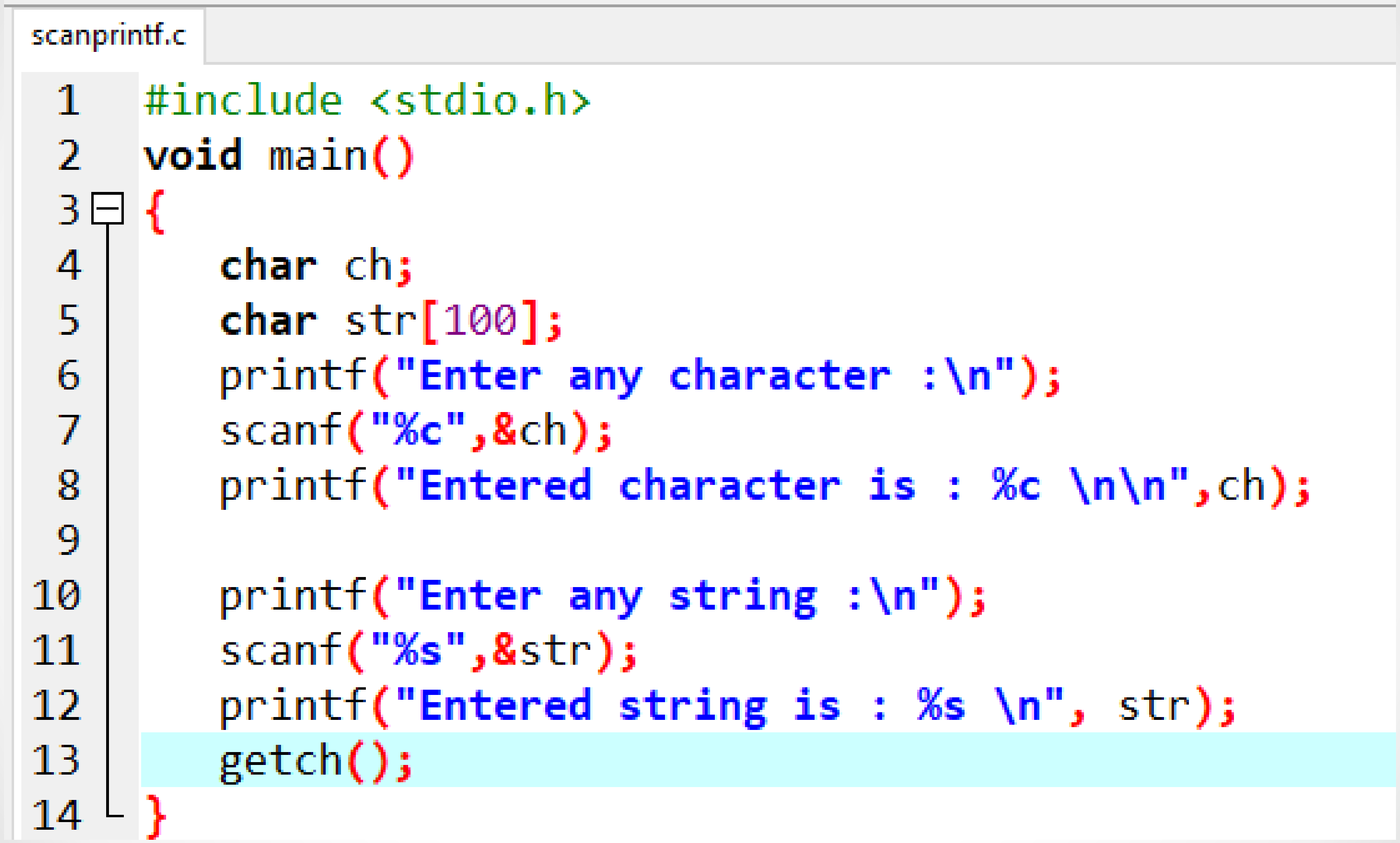
❖ **The *scanf* function**

- scanf() function are inbuilt library functions in C which are available in C library by default.
- **Input data can be entered into the computer from a standard input device** by means of the C library function scanf
- This function can be used to enter any combination of numerical values, single characters and strings
- **scanf() function is used to read character, string, numeric data from keyboard.**
- Each character group must begin with a percent sign (%), followed by a *conversion character* which indicates the type of the corresponding data item (The format specifier %d is used in scanf() statement for integer)
- The arguments are written as variables , whose types match the corresponding character groups in the control string. Each variable name must be preceded by an ampersand (&) which will obtain the address of variable. ("ch" in scanf() statement as &ch)

➢ The ***printf*** function

- printf() function are inbuilt library functions in C which are available in C library by default
- **Output data can be written from the computer onto a standard output device using the library function printf.**
- This function can be used to output any combination of numerical values, single characters and strings
- It is similar to the input function scanf, except that its **purpose is to display data** rather than to enter it into the computer
- The control string consists of individual groups of characters, with one character group for each output data item. Each character group must begin with a percent sign (%).
- The arguments in a printf function **do not represent memory addresses and therefore are not preceded by ampersands.**

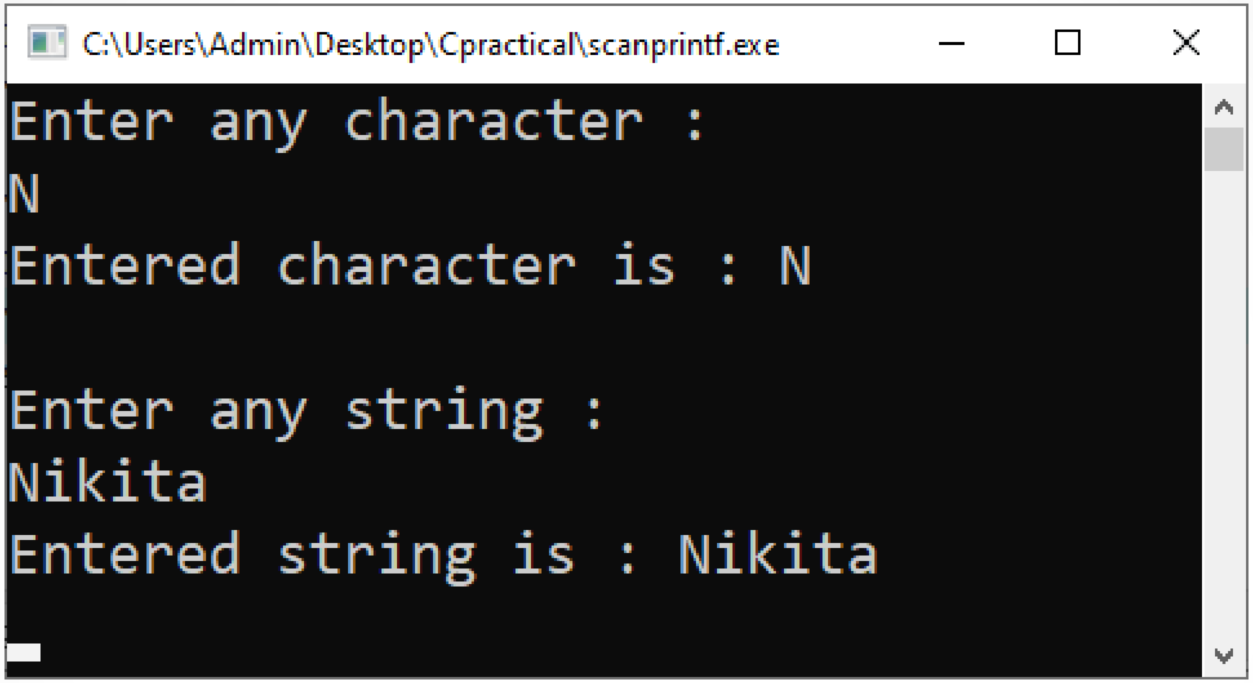
e.g.



scanprintf.c

```c
#include <stdio.h>
void main()
{
    char ch;
    char str[100];
    printf("Enter any character :\n");
    scanf("%c",&ch);
    printf("Entered character is : %c \n\n",ch);

    printf("Enter any string :\n");
    scanf("%s",&str);
    printf("Entered string is : %s \n", str);
    getch();
}
```

Output :


C:\Users\Admin\Desktop\Cpractical\scanprintf.exe

```
Enter any character :
N
Entered character is : N

Enter any string :
Nikita
Entered string is : Nikita
```