CSE 06131223 ♦ CSE 06131224

Structured Programming

Lecture 16

Functions in C (3)

Prepared by

Md. Mijanur Rahman, Prof. Dr.
Dept. of Computer Science and Engineering
Jatiya Kabi Kazi Nazrul Islam University, Bangladesh
www.mijanrahman.com

Contents

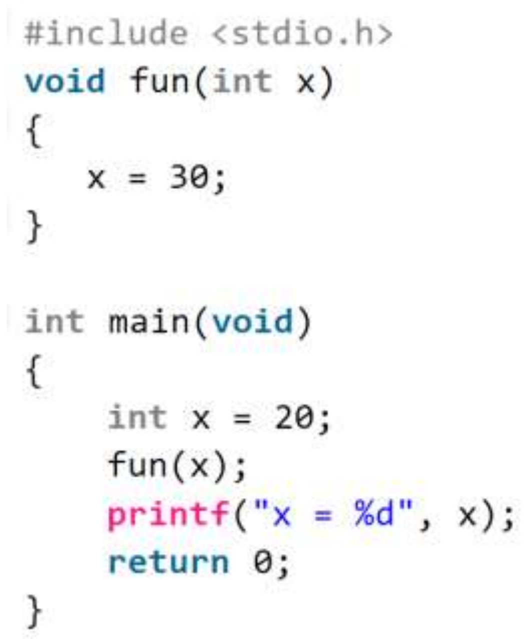
FUNCTIONS IN C

Parameter Passing to Functions

- The parameters passed to function are called ***actual parameters***. The parameters received by function are called ***formal parameters***.
- There are two most popular ways to pass parameters.
- ***Pass by Value:*** In this parameter passing method, values of actual parameters are copied to function's formal parameters and the two types of parameters are stored in different memory locations. So any changes made inside functions are not reflected in actual parameters of caller.
- ***Pass by Reference*** Both actual and formal parameters refer to same locations, so any changes made inside the function are actually reflected in actual parameters of caller.

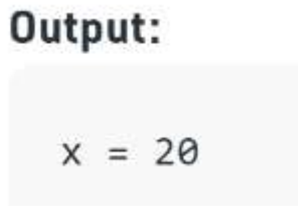
Parameter Passing to Functions

- **Pass by value:**
  - Parameters are always passed by value in C. For example. in the below code, value of x is not modified using the function fun().



```c
#include <stdio.h>
void fun(int x)
{
   x = 30;
}

int main(void)
{
     int x = 20;
     fun(x);
     printf("x = %d", x);
     return 0;
}
```

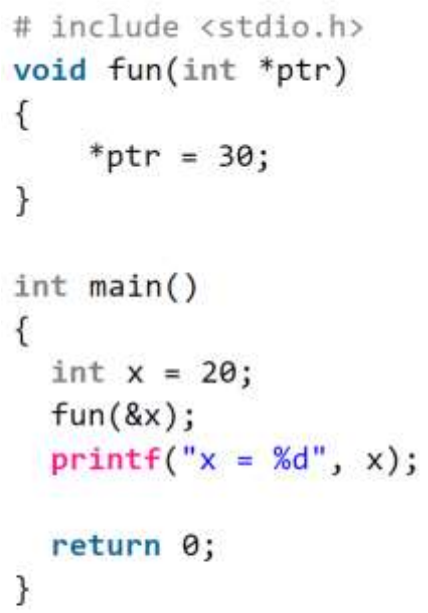


Output:

```
x = 20
```

Parameter Passing to Functions

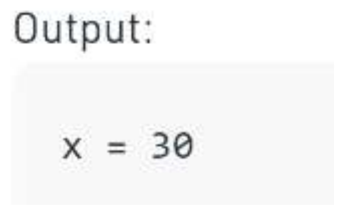
- **Pass by reference:**
  - In C, we can use pointers to get the effect of pass-by reference. For example, consider the below program. The function **fun()** expects a pointer **ptr** to an integer (or an address of an integer). It modifies the value at the address **ptr**. The address operator **&** is used to get the address of a variable of any data type.



```
# include <stdio.h>
void fun(int *ptr)
{
    *ptr = 30;
}

int main()
{
  int x = 20;
  fun(&x);
  printf("x = %d", x);

  return 0;
}
```



```
Output:

  x = 30
```

Some Important Point About C Functions

- **Following are some important points about functions in C:**
  1. Every C program has a function called main() that is called by operating system when a user runs the program.
  2. Every function has a return type. If a function doesn't return any value, then void is used as a return type.
  3. In C, functions can return any type except arrays and functions. We can get around this limitation by returning pointer to array or pointer to function.
  4. Empty parameter list in C means that the parameter list is not specified and function can be called with any parameters. In C, it is not a good idea to declare a function like fun(). To declare a function that can only be called without any parameter, we should use "void fun(void)".
  5. If in a C program, a function is called before its declaration then the C compiler automatically assumes the declaration of that function in the following way:

     ```
     int function name();
     ```

Main Function

- The main function is a special function. Every C++ program must contain a function named main. It serves as the entry point for the program. The computer will start running the code from the beginning of the main function.
- **Types of main Function:**

1. The first type is – main function without parameters :

```
// Without Parameters
int main()
{
    ...
    return 0;
}
```

Main Function

2. Second type is main function with parameters :

```
// With Parameters
int main(int argc, char* const argv[])
{
    ...
    return 0;
}
```

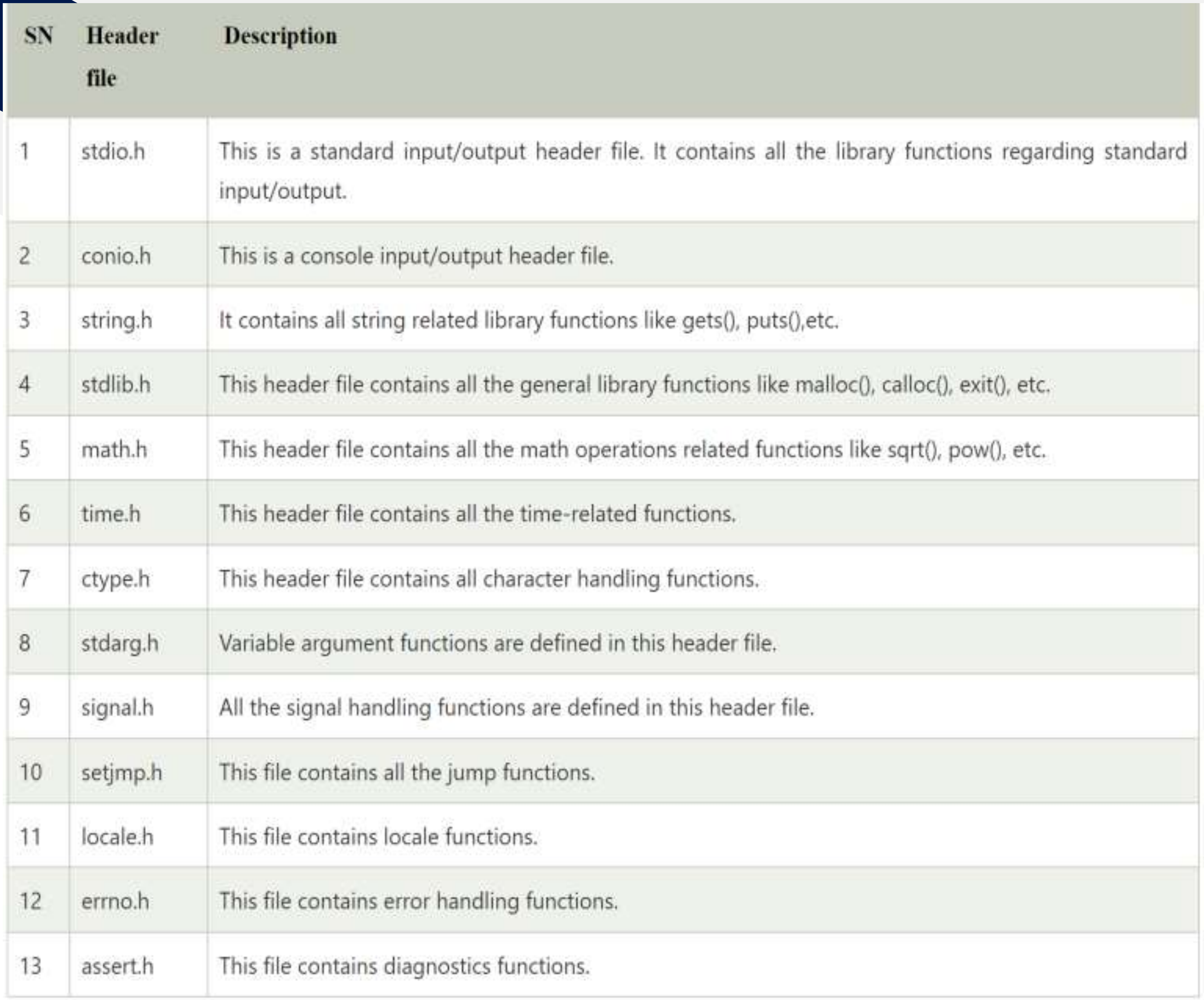
- The reason for having the parameter option for the main function is to allow input from the command line.

C Library Functions

- Library functions are the inbuilt function in C that are grouped and placed at a common place called the library. Such functions are used to perform some specific operations.
- For example, **printf** is a library function used to print on the console. The library functions are created by the designers of compilers.
- All C standard library functions are defined inside the different header files saved with the extension **.h**.
- We need to include these header files in our program to make use of the library functions defined in such header files.
- For example, To use the library functions such as **printf/scanf** we need to include **stdio.h** in our program which is a header file that contains all the library functions regarding standard input/output.

C Library Functions

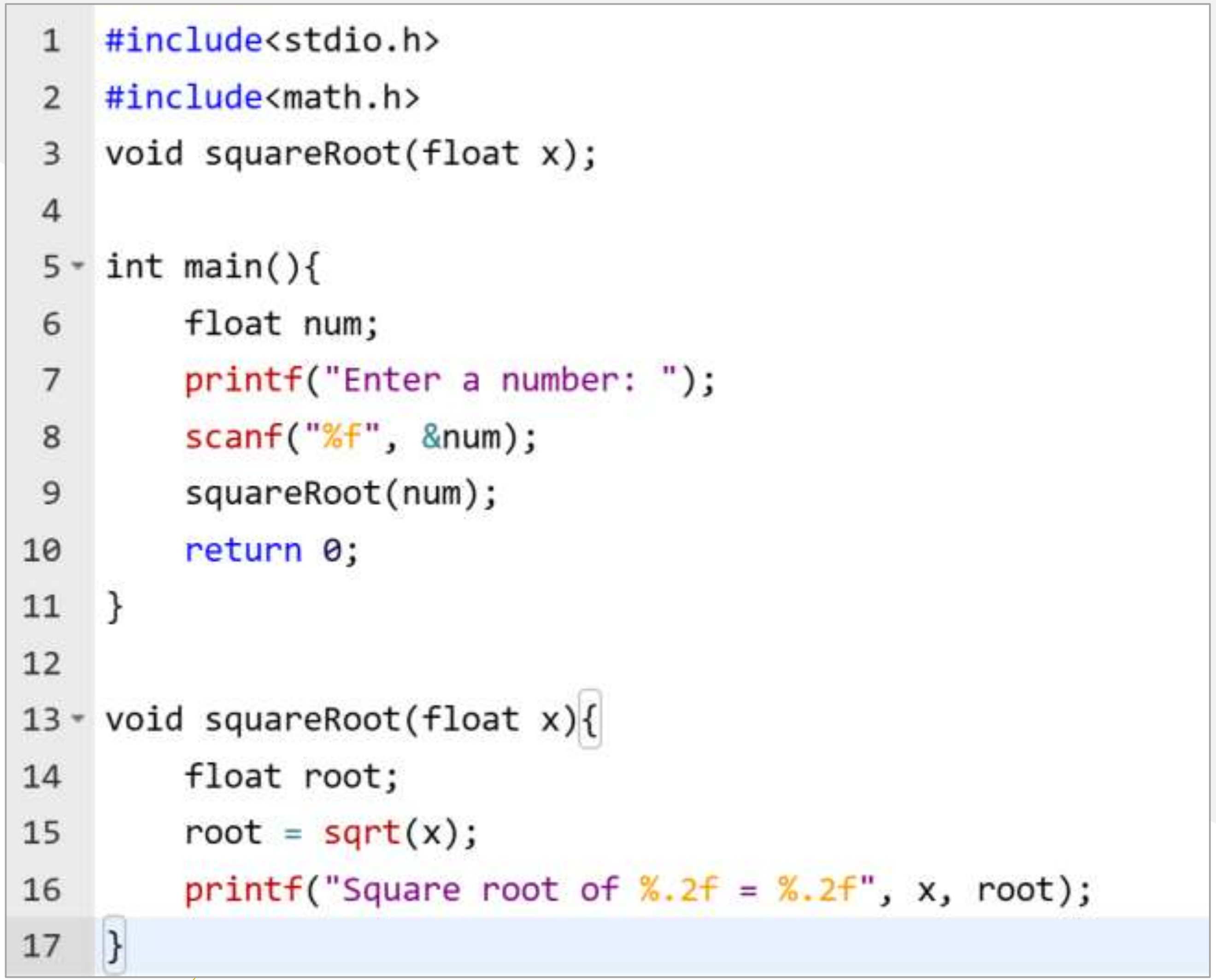
- **List of mostly used header files:**



SN	Header file	Description
1	stdio.h	This is a standard input/output header file. It contains all the library functions regarding standard input/output.
2	conio.h	This is a console input/output header file.
3	string.h	It contains all string related library functions like gets(), puts(),etc.
4	stdlib.h	This header file contains all the general library functions like malloc(), calloc(), exit(), etc.
5	math.h	This header file contains all the math operations related functions like sqrt(), pow(), etc.
6	time.h	This header file contains all the time-related functions.
7	ctype.h	This header file contains all character handling functions.
8	stdarg.h	Variable argument functions are defined in this header file.
9	signal.h	All the signal handling functions are defined in this header file.
10	setjmp.h	This file contains all the jump functions.
11	locale.h	This file contains locale functions.
12	errno.h	This file contains error handling functions.
13	assert.h	This file contains diagnostics functions.

Example

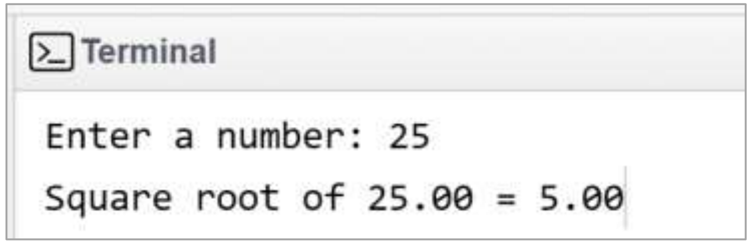
- **Program to calculate the square root of a number using function.**



```c
#include<stdio.h>
#include<math.h>
void squareRoot(float x);

int main(){
    float num;
    printf("Enter a number: ");
    scanf("%f", &num);
    squareRoot(num);
    return 0;
}

void squareRoot(float x){
    float root;
    root = sqrt(x);
    printf("Square root of %.2f = %.2f", x, root);
}
```



```
Enter a number: 25
Square root of 25.00 = 5.00
```

Creating Library Functions

- **STEPS FOR ADDING OUR OWN FUNCTIONS IN C LIBRARY:**

STEP 1:

- For example, below is a sample function that is going to be added in the C library. Write the below function in a file and save it as “addition.c”:

```
addition(int i, int j)
{
int total;
total = i + j;
return total;
}
```

STEP 3:

- “addition.obj” file would be created which is the compiled form of “addition.c” file.

Creating Library Functions

STEP 4:

- Use the below command to add this function to library (in turbo C).
  c:\> tlib math.lib + c:\ addition.obj

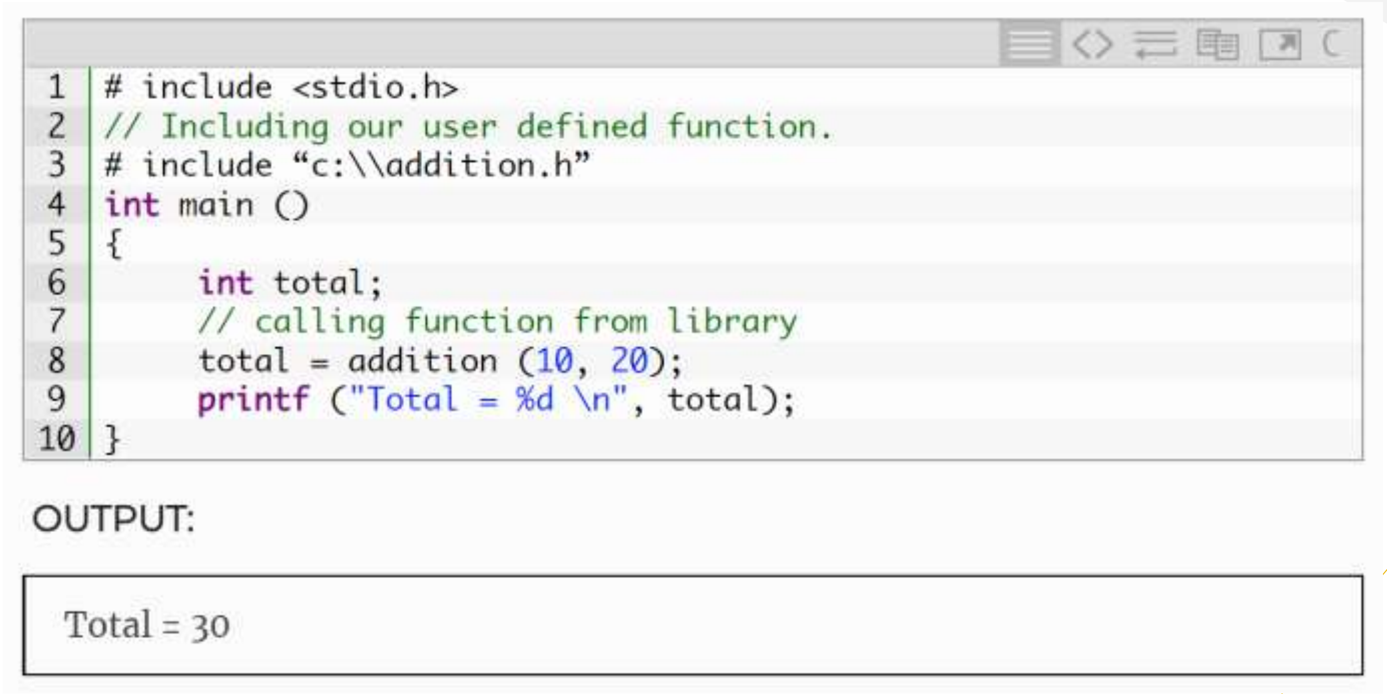
  + means adding c:\addition.obj file in the math library.
  We can delete this file using – (minus).

STEP 5:

- Create a file “addition.h” & declare prototype of addition() function like below.
  int addition (int i, int j);
- Now, addition.h file contains prototype of the function “addition”.
- Note : Please create, compile and add files in the respective directory as directory name may change for each IDE.

Creating Library Functions

- **STEP 6:**
- Let us see how to use our newly added library function in a C program.



```
# include <stdio.h>
// Including our user defined function.
# include “c:\\addition.h”
int main ()
{
      int total;
      // calling function from library
      total = addition (10, 20);
      printf ("Total = %d \n", total);
}
```

OUTPUT:

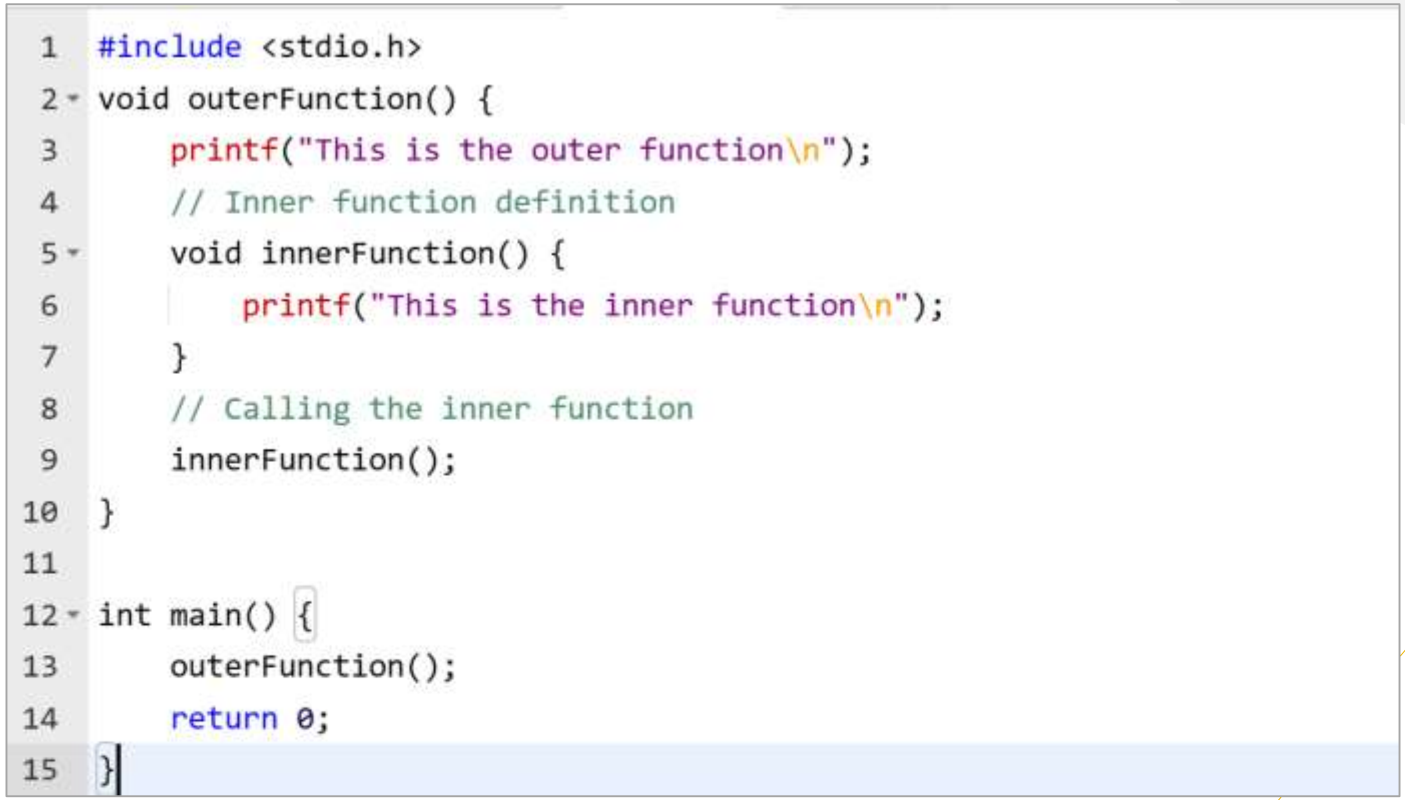
```
Total = 30
```

Nesting of Functions

- In C programming, nesting of functions refers to the practice of defining a function inside another function.
- This concept is also known as "nested functions" or "inner functions". When a function is defined within another function, the inner function has access to the variables and parameters of the outer function.
- However, the outer function cannot access the variables or parameters of the inner function.
- C permits nesting of functions freely. The main() calls function1, which calls function2, which calls function3, …., and so on. There is in principle no limit as to how deeply functions can be nested.

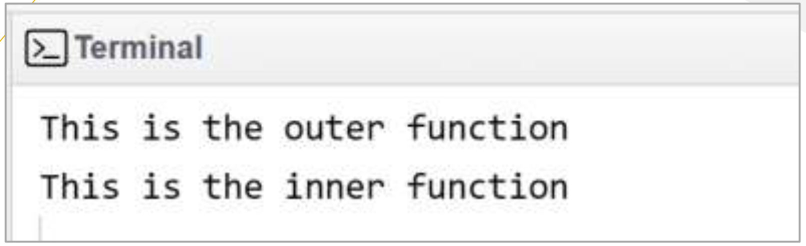
Nesting of Functions

- A simple example to illustrate nesting of functions in C:



```c
#include <stdio.h>
void outerFunction() {
    printf("This is the outer function\n");
    // Inner function definition
    void innerFunction() {
        printf("This is the inner function\n");
    }
    // Calling the inner function
    innerFunction();
}

int main() {
    outerFunction();
    return 0;
}
```

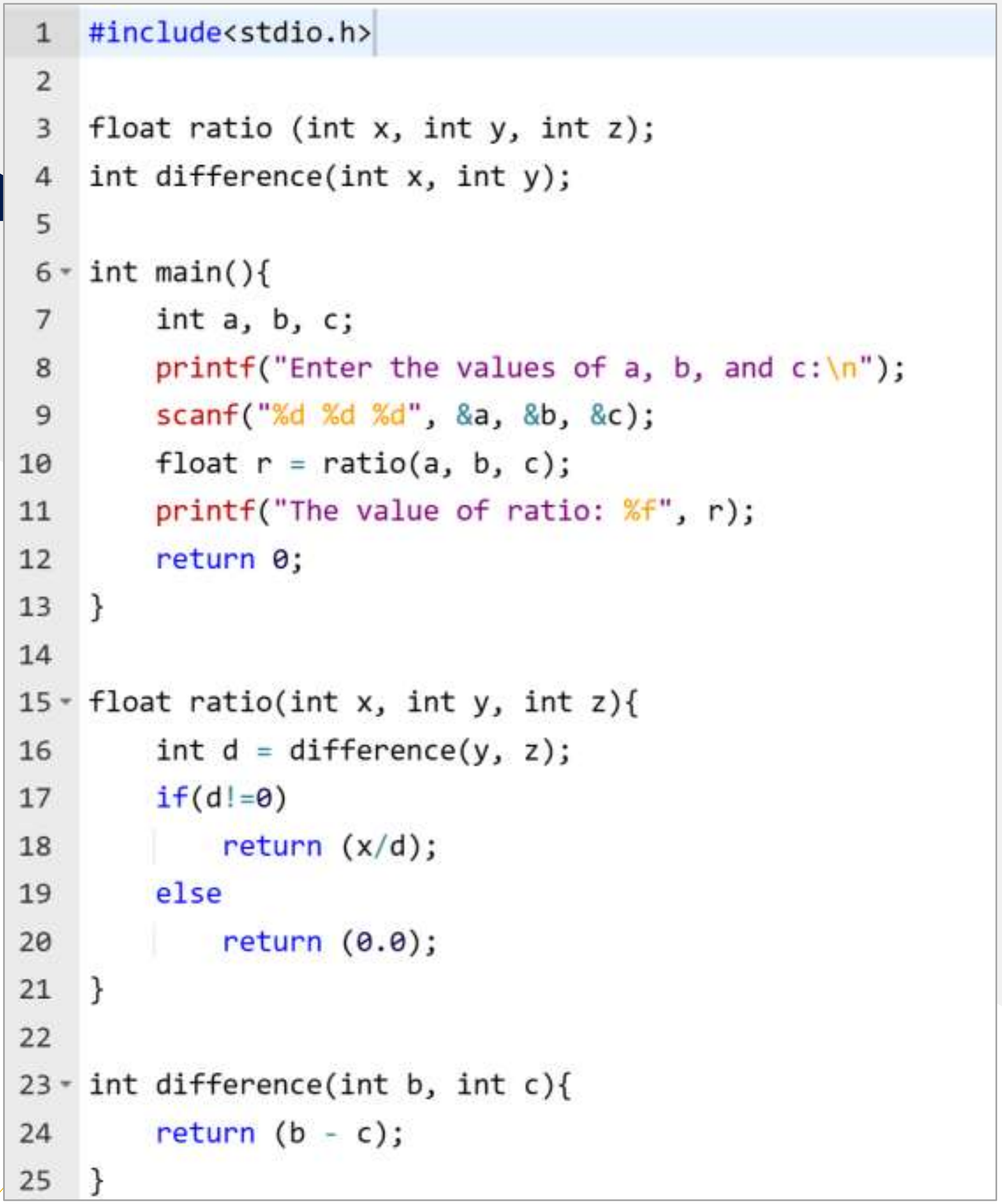


```
This is the outer function
This is the inner function
```

Nesting of Functions

- Consider the following ratio and write a program to calculate the ratio:

$$r = \frac{x}{y-z}$$



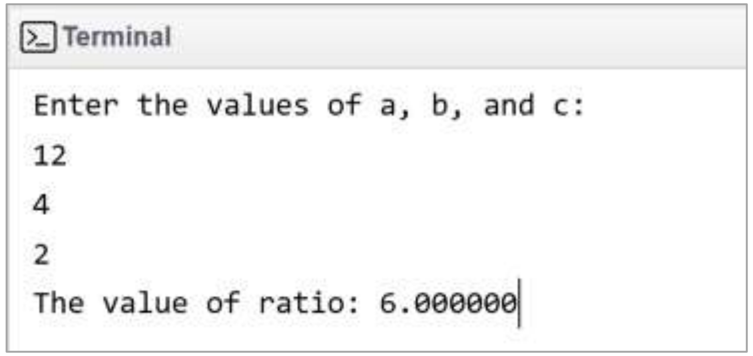
```c
#include<stdio.h>

float ratio (int x, int y, int z);
int difference(int x, int y);

int main(){
    int a, b, c;
    printf("Enter the values of a, b, and c:\n");
    scanf("%d %d %d", &a, &b, &c);
    float r = ratio(a, b, c);
    printf("The value of ratio: %f", r);
    return 0;
}

float ratio(int x, int y, int z){
    int d = difference(y, z);
    if(d!=0)
        return (x/d);
    else
        return (0.0);
}

int difference(int b, int c){
    return (b - c);
}
```



```
Enter the values of a, b, and c:
12
4
2
The value of ratio: 6.000000
```

Recursion

- Recursion in C refers to the process where a function calls itself directly or indirectly. It's a powerful concept in programming where a problem is solved by dividing it into smaller sub-problems of the same type.
- Recursion is widely used to solve problems that can be broken down into smaller, similar sub-problems.
- Recursion has a few key components:
  1. **Base Case:** This is the condition under which the function stops calling itself and returns a result. Without a base case, the recursion would continue indefinitely, leading to a stack overflow.
  2. **Recursive Case:** This is where the function calls itself with a smaller input, getting closer to the base case.
  3. **Stopping Condition:** The recursive calls must eventually reach the base case. If not, the recursion will continue indefinitely.

Recursion

- An useful example of recursion is the evaluation of factorials of a given number. The factorial of a number n is expressed as a series of repetitive multiplications as given below:
- Factorial of n = n(n-1)(n-2)…..1
- For examples, factorial of 4 = 4x3x2x1 = 24
- Thus, fact of n = n * factorial (n-1)

```
As n = 3,      fact      = 3 * factorial (2)
                         = 3 * 2 * factorial (1)
                         = 3 * 2 * 1
                         = 6
```

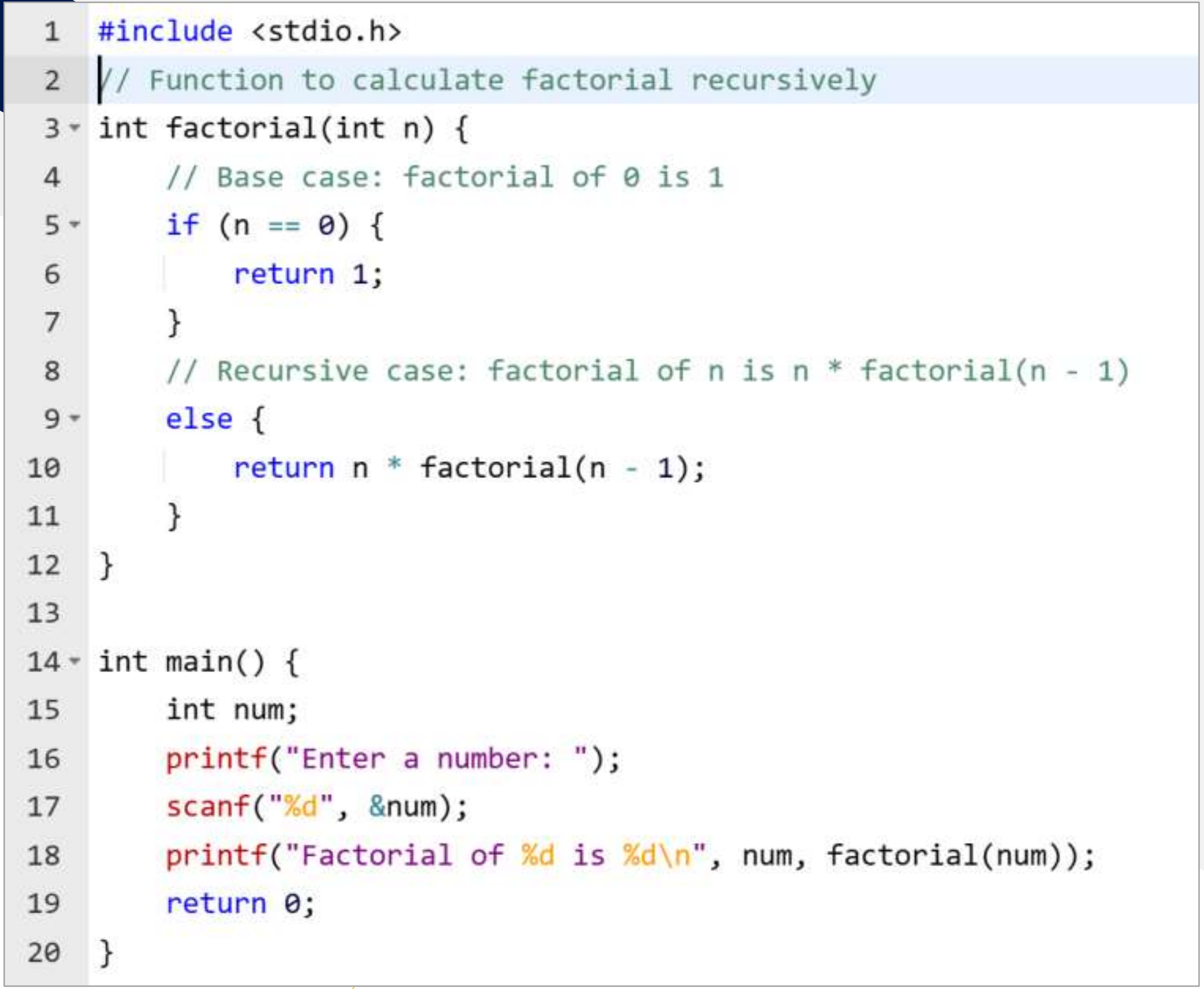
Recursion

- A function to evaluate factorial of n is as follows:
- Recursive Function factorial:

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

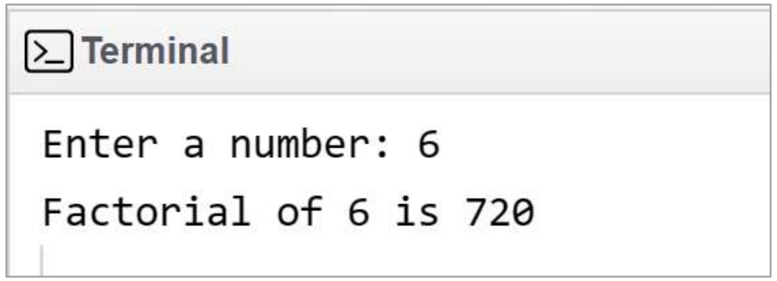
Recursion

- A function to evaluate factorial of n:



```c
#include <stdio.h>
// Function to calculate factorial recursively
int factorial(int n) {
    // Base case: factorial of 0 is 1
    if (n == 0) {
        return 1;
    }
    // Recursive case: factorial of n is n * factorial(n - 1)
    else {
        return n * factorial(n - 1);
    }
}

int main() {
    int num;
    printf("Enter a number: ");
    scanf("%d", &num);
    printf("Factorial of %d is %d\n", num, factorial(num));
    return 0;
}
```



```
Enter a number: 6
Factorial of 6 is 720
```

Recursion

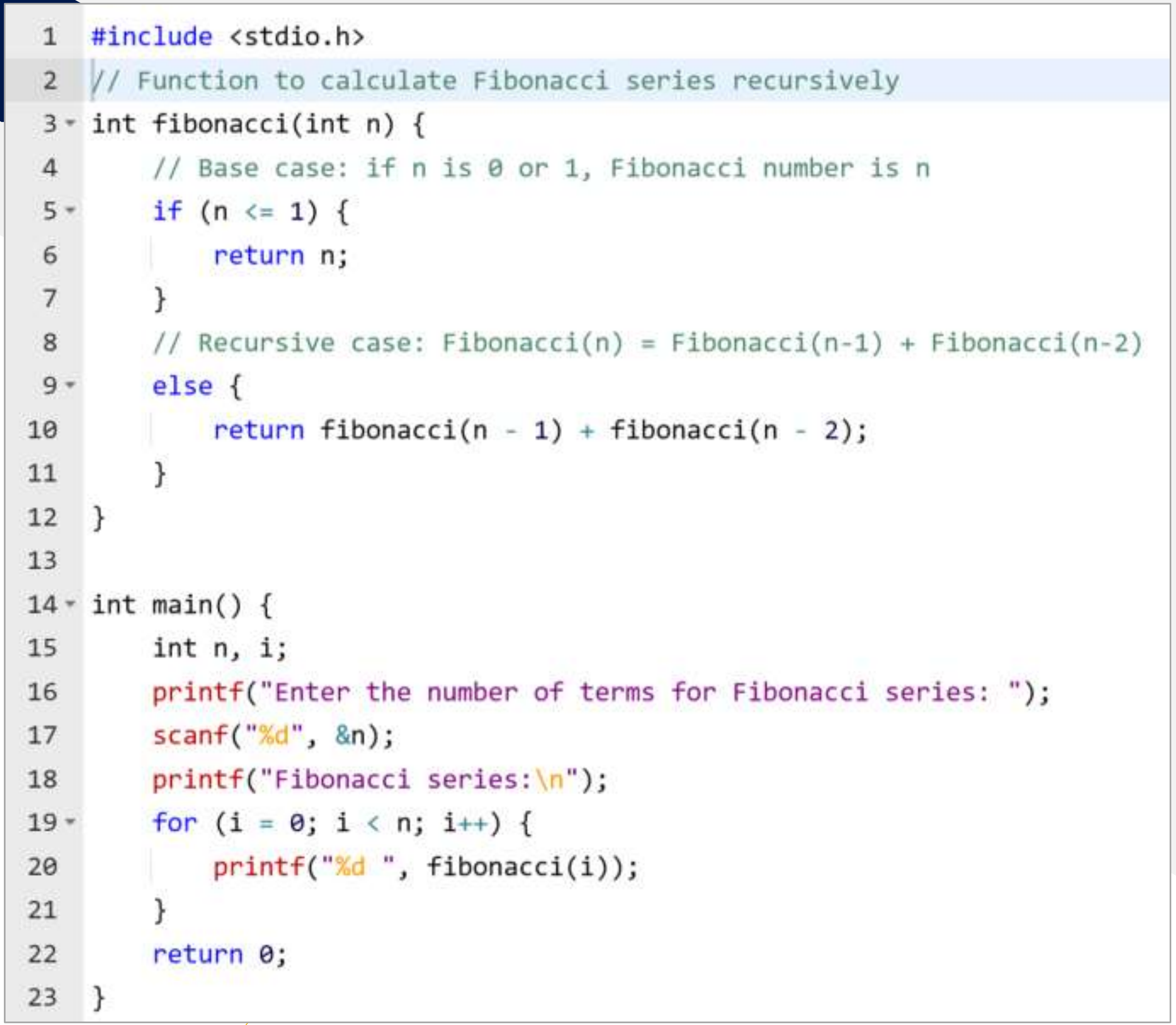
- Recursion function to find Fibonacci series: The Fibonacci series is a sequence of numbers where each number is the sum of the two preceding ones, usually starting with 0 and 1. So, the Fibonacci sequence starts as 0, 1, 1, 2, 3, 5, 8, 13, 21, and so on.
- **Recursive Function Fibonacci:**

```
int fibonacci(int n) {
        if (n <= 1)
           return n;
        else
          return fibonacci(n - 1) + fibonacci(n - 2);
 }
```

- Base case: The base case is the simplest form of the problem that can be solved directly without further recursion. In this case, if n is 0 or 1, the Fibonacci number is n itself. So, if n is less than or equal to 1, the function returns n.
- Recursive case: If n is greater than 1, the Fibonacci number for n is the sum of the Fibonacci numbers for n-1 and n-2. So, the function recursively calls itself with n-1 and n-2, adds the results, and returns the sum.

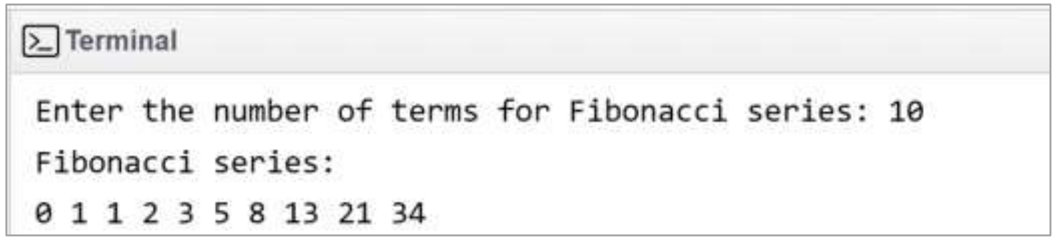
Recursion

- **Recursion function to find Fibonacci series:**



```c
#include <stdio.h>
// Function to calculate Fibonacci series recursively
int fibonacci(int n) {
    // Base case: if n is 0 or 1, Fibonacci number is n
    if (n <= 1) {
        return n;
    }
    // Recursive case: Fibonacci(n) = Fibonacci(n-1) + Fibonacci(n-2)
    else {
        return fibonacci(n - 1) + fibonacci(n - 2);
    }
}

int main() {
    int n, i;
    printf("Enter the number of terms for Fibonacci series: ");
    scanf("%d", &n);
    printf("Fibonacci series:\n");
    for (i = 0; i < n; i++) {
        printf("%d ", fibonacci(i));
    }
    return 0;
}
```



```
Enter the number of terms for Fibonacci series: 10
Fibonacci series:
0 1 1 2 3 5 8 13 21 34
```

Assignment and Presentation On

- **The Scope, Visibility and Lifetime of Variables:**
  - Automatic Variables
  - External Variables
  - Static Variables
  - Register Variables

- Explain each terms with C programs.

? THE END