# FUNCTION

## A - Theory - References

A function is a block of code that performs a specific task.

Suppose, you need to create a program to create a circle and color it. You can create two functions to solve this problem:

* create a circle function
* create a color function

Dividing a complex problem into smaller chunks makes our program easy to understand and reuse.

### 1. Types of function

There are two types of function in C programming:

* Standard library functions (Built-in)
* User-defined functions

#### a. Standard library functions

The standard library functions are built-in functions in C programming.

These functions are defined in header files. For example,

* The printf() is a standard library function to send formatted output to the screen (display output on the screen). This function is defined in the stdio.h header file.  
  Hence, to use the printf() function, we need to include the stdio.h header file using #include <stdio.h>.
* The sqrt() function calculates the square root of a number. The function is defined in the math.h header file.

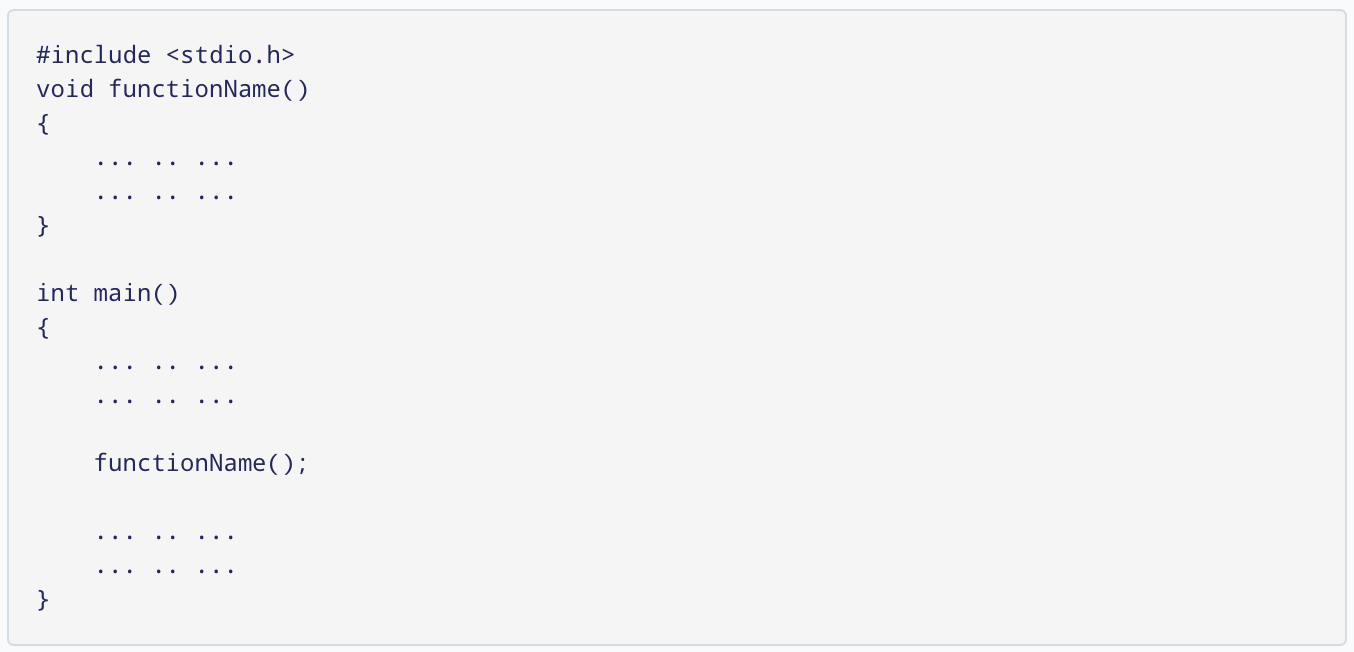
Library Functions in Different Header Files

C Header Files 
cassert.h> 
ectype.h> 
(locale.h> 
<stdlib. h) 
Program assertion functions 
Character type functions 
Localization functions 
Mathematics functions 
Jump functions 
Signal handling functions 
Variable arguments handling functions 
Standard Input/Output functions 
Standard utility functions 
String handling functions 
Date time functions 

#### b. User-defined function

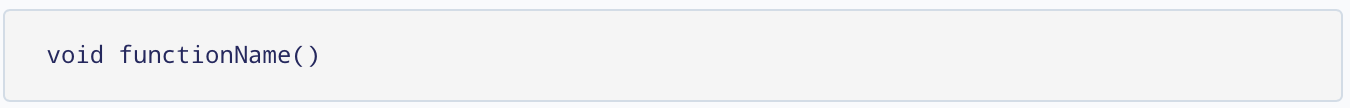
You can also create functions as per your need. Such functions created by the user are known as user-defined functions.

**How user-defined function works?**



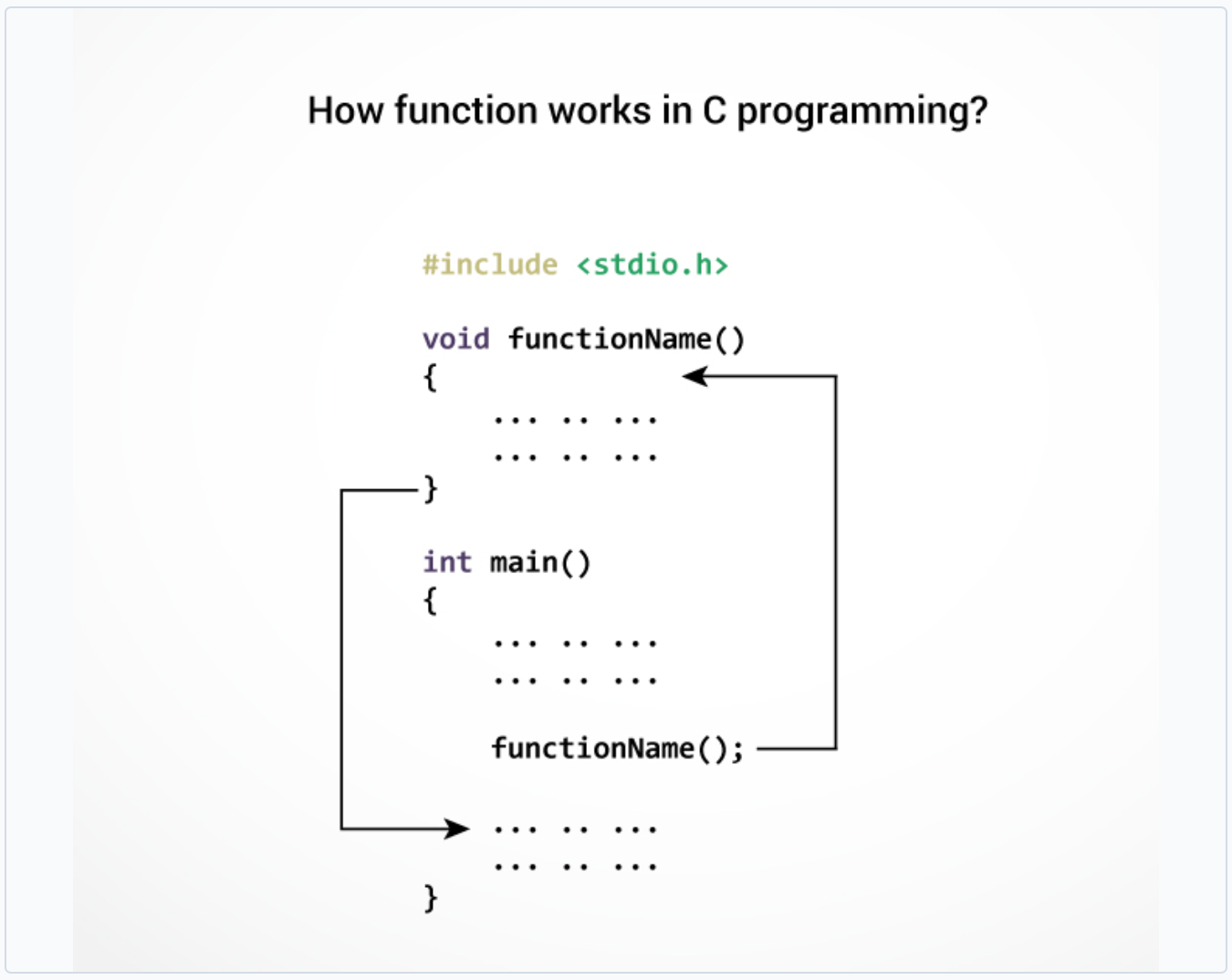
The execution of a C program begins from the  main()  function.

When the compiler encounters  functionName();, control of the program jumps to



And, the compiler starts executing the codes inside functionName().

The control of the program jumps back to the main() function once code inside the function definition is executed.



Note, function names are identifiers and should be unique.

**Advantages of user-defined function**

* The program will be easier to understand, maintain and debug.
* Reusable codes that can be used in other programs
* A large program can be divided into smaller modules. Hence, a large project can be divided among many programmers.

### 2. C User-defined functions

Here is an example to add two integers. To perform this task, we have created an user-defined addNumbers().

#include <stdio.h>

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

int main()

{

int n1,n2,sum;

printf("Enters two numbers: ");

scanf("%d %d",&n1,&n2);

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

printf("sum = %d",sum);

return 0;

}

int addNumbers(int a, int b) // function definition

{

int result;

result = a+b;

return result; // return statement

}

------------------------------------------------------------------------------------------------------------------

#### a. Function prototype

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.

A function prototype gives information to the compiler that the function may later be used in the program.

#### b. Syntax of function prototype

returnType functionName(type1 argument1, type2 argument2, ...);

In the above example, int addNumbers(int a, int b); is the function prototype which provides the following information to the compiler:

* name of the function is addNumbers()
* return type of the function is int
* two arguments of type int are passed to the function

The function prototype is not needed if the user-defined function is defined before the main() function.

------------------------------------------------------------------------------------------------------------------

#### c. Calling a function

Control of the program is transferred to the user-defined function by calling it.

*Syntax of function call*

functionName(argument1, argument2, ...);

In the above example, the function call is made using addNumbers(n1, n2); statement inside the main() function.

-------------------------------------------------------------------------------------------------------------

#### d. Function definition

Function definition contains the block of code to perform a specific task. In our example, adding two numbers and returning it.

*Syntax of function definition*

returnType functionName(type1 argument1, type2 argument2, ...)

{

//body of the function

}

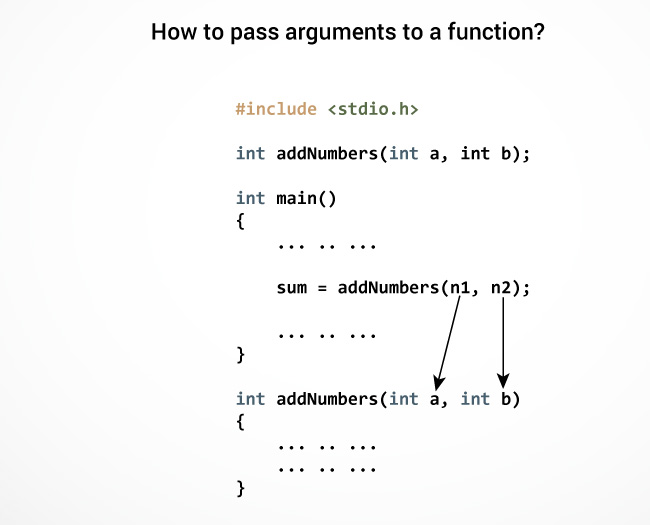
When a function is called, the control of the program is transferred to the function definition. And, the compiler starts executing the codes inside the body of a function.

------------------------------------------------------------------------------------------------------------------

#### e. Passing arguments to a function

In programming, argument refers to the variable passed to the function. In the above example, two variables n1 and n2 are passed during the function call.

The parameters a and b accepts the passed arguments in the function definition. These arguments are called formal parameters of the function.



The type of arguments passed to a function and the formal parameters must match, otherwise, the compiler will throw an error.

If n1 is of char type, a also should be of char type. If n2 is of float type, variable b also should be of float type.

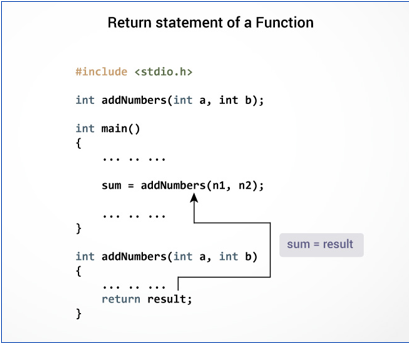
A function can also be called without passing an argument.

------------------------------------------------------------------------------------------------------------------

#### f. Return Statement

The return statement terminates the execution of a function and returns a value to the calling function. The program control is transferred to the calling function after the return statement.

In the above example, the value of the result variable is returned to the main function. The sum variable in the main() function is assigned this value.



#### g. Syntax of return statement

return (expression);

For example,

return a;

return (a+b);

The type of value returned from the function and the return type specified in the function prototype and function definition must match.

### 3. Types of User-defined Functions in C Programming

These 4 programs below check whether the integer entered by the user is a prime number or not.

The output of all these programs below is the same, and we have created a user-defined function in each example. However, the approach we have taken in each example is different.

**Example 1: No arguments passed and no return value**

#include <stdio.h>

void checkPrimeNumber();

int main()

{

checkPrimeNumber(); // argument is not passed

return 0;

}

// return type is void meaning doesn't return any value

void checkPrimeNumber()

{

int n, i, flag = 0;

printf("Enter a positive integer: ");

scanf("%d",&n);

for(i=2; i <= n/2; ++i)

{

if(n%i == 0)

{

flag = 1;

}

}

if (flag == 1)

printf("%d is not a prime number.", n);

else

printf("%d is a prime number.", n);

}

The checkPrimeNumber() function takes input from the user, checks whether it is a prime number or not and displays it on the screen.

The empty parentheses in checkPrimeNumber(); statement inside the main() function indicates that no argument is passed to the function.

The return type of the function is void. Hence, no value is returned from the function.

**Example 2: No arguments passed but a return value**

#include <stdio.h>

int getInteger();

int main()

{

int n, i, flag = 0;

// no argument is passed

n = getInteger();

for(i=2; i<=n/2; ++i)

{

if(n%i==0){

flag = 1;

break;

}

}

if (flag == 1)

printf("%d is not a prime number.", n);

else

printf("%d is a prime number.", n);

return 0;

}

// returns integer entered by the user

int getInteger()

{

int n;

printf("Enter a positive integer: ");

scanf("%d",&n);

return n;

}

The empty parentheses in the n = getInteger(); statement indicates that no argument is passed to the function. And, the value returned from the function is assigned to n.

Here, the getInteger() function takes input from the user and returns it. The code to check whether a number is prime or not is inside the main() function.

**Example 3: Argument passed but no return value**

#include <stdio.h>

void checkPrimeAndDisplay(int n);

int main()

{

int n;

printf("Enter a positive integer: ");

scanf("%d",&n);

// n is passed to the function

checkPrimeAndDisplay(n);

return 0;

}

// return type is void meaning doesn't return any value

void checkPrimeAndDisplay(int n)

{

int i, flag = 0;

for(i=2; i <= n/2; ++i)

{

if(n%i == 0){

flag = 1;

break;

}

}

if(flag == 1)

printf("%d is not a prime number.",n);

else

printf("%d is a prime number.", n);

}

The integer value entered by the user is passed to the checkPrimeAndDisplay() function.

Here, the checkPrimeAndDisplay() function checks whether the argument passed is a prime number or not and displays the appropriate message.

**Example 4: Argument passed and a return value**

#include <stdio.h>

int checkPrimeNumber(int n);

int main()

{

int n, flag;

printf("Enter a positive integer: ");

scanf("%d",&n);

// n is passed to the checkPrimeNumber() function

// the returned value is assigned to the flag variable

flag = checkPrimeNumber(n);

if(flag == 1)

printf("%d is not a prime number",n);

else

printf("%d is a prime number",n);

return 0;

}

// int is returned from the function

int checkPrimeNumber(int n)

{

int i;

for(i=2; i <= n/2; ++i)

{

if(n%i == 0)

return 1;

}

return 0;

}

The input from the user is passed to the checkPrimeNumber() function.

The checkPrimeNumber() function checks whether the passed argument is prime or not.

If the passed argument is a prime number, the function returns 0. If the passed argument is a non-prime number, the function returns 1. The return value is assigned to the flag variable.

Depending on whether flag is 0 or 1, an appropriate message is printed from the main()function.

**Which approach is better?**

Well, it depends on the problem you are trying to solve. In this case, passing argument and returning a value from the function (example 4) is better.

A function should perform a specific task. The checkPrimeNumber() function doesn't take input from the user nor it displays the appropriate message. It only checks whether a number is prime or not.

## B - Example: Function - Iteration - Conditional

#include <stdio.h>

#include <conio.h>

// Khai bao cac nguyen mau 'Prototype' cua cac ham trong CT

void Add();

void Sub();

void Multi();

void Div();

int main(){

int choice;

while(1){

system("cls");

printf("--- BASIC CALCULATOR ---\n");

printf("1. Addition\n");

printf("2. Subtraction\n");

printf("3. Multiple\n");

printf("4. Division\n");

printf("5. Exit\n");

printf("Select options: ");

scanf("%d", &choice);

switch(choice){

case 1:

Add(); break;

case 2:

Sub(); break;

case 3:

Multi(); break;

case 4:

Div(); break;

case 5:

printf("Thank!");

exit(0);

break;

default:

printf("Invalid option. Please try again!");

break;

}

getch();

}

return 0;

}

// Tien hanh dinh nghia cac ham da duoc khai bao

void Add(){

int a, b, result;

printf("Enter a: "); scanf("%d", &a);

printf("Enter b: "); scanf("%d", &b);

result = a + b;

printf("Total a + b = %d\n", result);

}

void Sub(){

int a, b, result;

printf("Enter a: "); scanf("%d", &a);

printf("Enter b: "); scanf("%d", &b);

result = a - b;

printf("Sub a - b = %d\n", result);

}

void Multi(){

int a, b, result;

printf("Enter a: "); scanf("%d", &a);

printf("Enter b: "); scanf("%d", &b);

result = a \* b;

printf("Multi a \* b = %d\n", result);

}

void Div(){

int a, b;

double result;

printf("Enter a: "); scanf("%d", &a);

printf("Enter b: "); scanf("%d", &b);

result = (double)a / b;

printf("Div a / b = %.2lf\n", result);

}

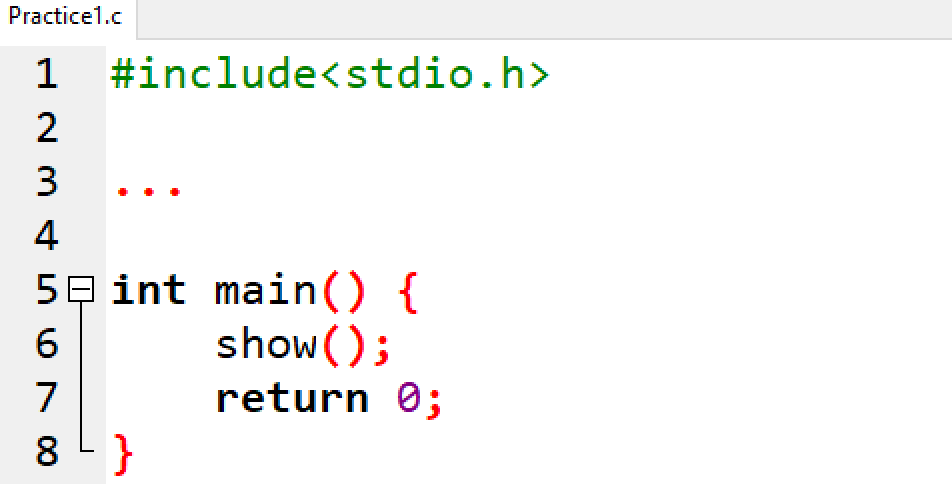
## C - PRACTICE

**P1. Write a function that prints all numbers from 1 to 100 as below:**



-----

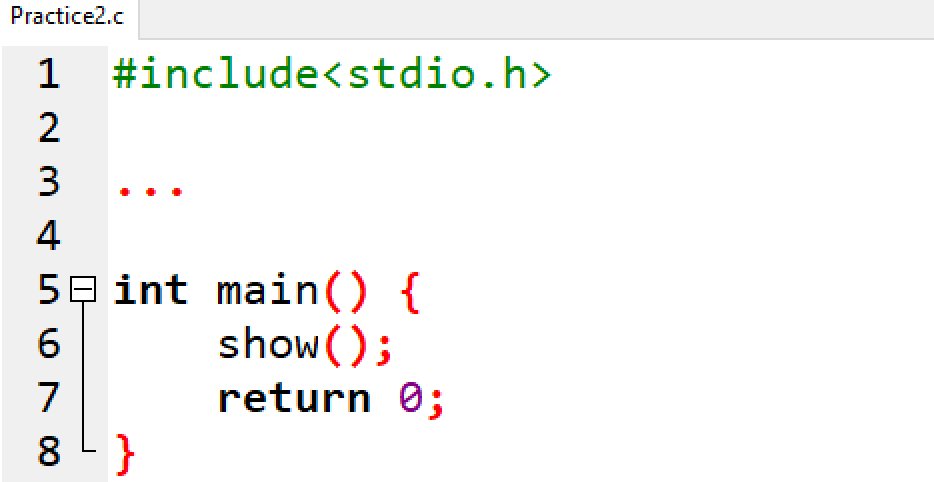
**Program**:



**P2. Write a function to print all the numbers from 1 to 100, which are divisible by 3 and not divisible by 5.**

-----

**Program**:



**P3. Write a function to find all the prime numbers between the two integers, checkPrimeNumber() is a function created. A prime number is a positive integer that is divisible only by 1 and itself. For example: 2, 3, 5, 7, 11, 13, 17**

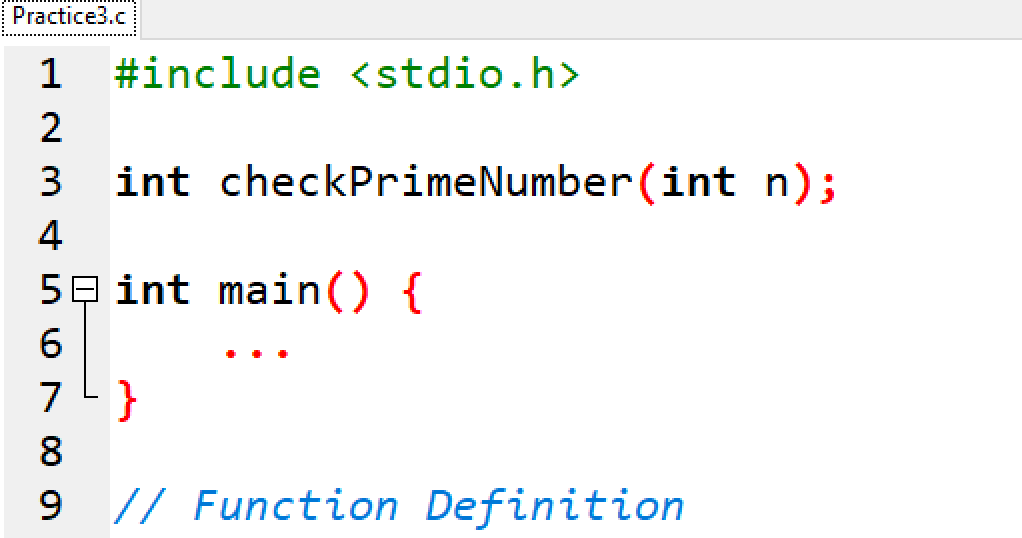
**Output**:



(Note: If the user enters the larger number first, this program will not work as intended. To solve this issue, you need to swap the numbers first)

-----

**Program**:



**P4. Write a program in C to convert decimal number to binary number using the function.**

**Output**:

decimal number: 
19 
Enter a 
19/2, Remainder = 
Step 1: 
1, 
9/2, Remainder 
Step 2: 
4/2, Remainder 
Step 3: 
2/2, Remainder 
Step 4: 
1/2, Remainder 
Step 5: 
-1, 
Quotient 
Quotient 
Quotient 
Quotient 
Quotient 
19 in decimal 
10011 in 
binary 

-----

**Program**:

Practice4.c 
2 
3 
10 
11 
#include <stdio.h> 
long long convert (int n); 
int main 
int n; 
printf( "Enter a decimal number: " 
scanf( "%d", &n); 
printf( in decimal = %lld in binary", 
return 0; 
n, 
convert (n)); 

## D - EXERCISES

**Ex1. Write a program in C to find the sum of the series 1!/1 + 2!/2 + 3!/3 + 4!/4 + 5!/5 using the function.**

*Expected Output* :

The sum of the series is : 34

**Ex2. Write a program in C to check a given number is even or odd using the function.**

Test Data :

Input any number : 5

*Expected Output* :

The entered number is odd.

## E - REFERENCE

* Write a program that either calculates the value of an integer raised to the power of an integer exponent or the arithmetic mean of a series of integers, depending upon a choice made by the user.
  + list all of the tasks that the program should perform to solve this problem
  + identify the modules for the problem structure
  + check that each module is high in cohesion
  + check that each module is low in coupling

----

**Solution**:

#include <stdio.h>

// Prototypes

int power(int base, int exponent);

int sum(int number);

int main(){

char choice, confirm;

int number, exponent;

do{

system("cls");

printf("-- Exercises 1 --\n");

printf("1. Power of an integer exponent\n");

printf("2. Sum of a series of integers\n");

printf("3. Exit\n");

printf("Select a option: ");

scanf(" %c", &choice);

switch(choice){

case '1':

printf("Enter base: ");

scanf("%d", &number);

printf("Enter exponent: ");

scanf("%d", &exponent);

printf("%d^%d = %d", number, exponent, power(number,exponent));

break;

case '2':

printf("Enter a integer: ");

scanf("%d", &number);

printf("Sum: 1 + 2 + ... + %d = %d", number, sum(number));

break;

case '3':

printf("Goodbye!");

exit(0);

break;

default:

printf("Invalid option. Try again!");

break;

}

printf("\nPress y or Y to continue: ");

scanf(" %c", &confirm);

} while(confirm=='y' || confirm=='Y');

return 0;

}

// Defination functions

int power(int base, int exponent){

int result = 1, i;

for(i=0; i< exponent; i++)

result \*= base;

return result;

}

int sum(int number){

int result = 0, i;

for(i=0; i<=number; i++)

result += i;

return result;

}