UNIT 5 PROBLRM SOLVING USING C

**User-defined Functions**

A function is a group of statements that together perform a task. Every C program has at least one function, which is **main()**, and all the most trivial programs can define additional functions.

You can divide up your code into separate functions. How you divide up your code among different functions is up to you, but logically the division is such that each function performs a specific task.

A function **declaration** tells the compiler about a function's name, return type, and parameters. A function **definition** provides the actual body of the function.

The C standard library provides numerous built-in functions that your program can call. For example, **strcat()** to concatenate two strings, **memcpy()** to copy one memory location to another location, and many more functions.

A function can also be referred as a method or a sub-routine or a procedure, etc.

Defining a Function

The general form of a function definition in C programming language is as follows −

return\_type function\_name( parameter list ) {

body of the function

}

A function definition in C programming consists of a *function header* and a *function body*. Here are all the parts of a function −

* **Return Type** − A function may return a value. The **return\_type** is the data type of the value the function returns. Some functions perform the desired operations without returning a value. In this case, the return\_type is the keyword **void**.
* **Function Name** − This is the actual name of the function. The function name and the parameter list together constitute the function signature.
* **Parameters** − A parameter is like a placeholder. When a function is invoked, you pass a value to the parameter. This value is referred to as actual parameter or argument. The parameter list refers to the type, order, and number of the parameters of a function. Parameters are optional; that is, a function may contain no parameters.
* **Function Body** − The function body contains a collection of statements that define what the function does.

Example

Given below is the source code for a function called **max()**. This function takes two parameters num1 and num2 and returns the maximum value between the two −

/\* function returning the max between two numbers \*/

int max(int num1, int num2) {

/\* local variable declaration \*/

int result;

if (num1 > num2)

result = num1;

else

result = num2;

return result;

}

**Multifunction program :** A function is a self-contained block of code that performs a particular task. Once a function has been designed and packed, it can be treated as a ‘black box’ that takes some data from the main program and returns a value. Thus a program, which has been written using a number of functions, is treated as a multi-function program.

include<stdio.h>;

//defines PI at 3.14

#define PI 3.14

int main(void){

// five declared variables

int length = 0;

int width = 0;

int areaR = 0;

int radius = 0;

double areaC = 0.0;

//Ask the user the length and width of the rectangle

printf("Please enter the legnth and width of the rectangle: \n");

//Get the length and width from the user

scanf("%d%d", &length, &width);

//Calculate the area of the rectangle

areaR = length \* width;

//Print the area of the rectangle

printf("Area of the rectangle is: %d\n", areaR);

//Ask the useer for the radius of the circle

printf("Please enter the radius of the circle: \n");

//Get the radius of the circle

scanf("%d", &radius);

//Calculate and are of the circle

areaC = PI \* radius \* radius;

//Print the area of the circle

printf("The area of the circle is: %lf\n", areaC); // %lf is for double

return 0;

}

**Elements of User Defined Functions**

In order to make use of an user defined function, we need to establish three elements, which are as follows

1. Declaration
2. Definition
3. call

**Function Declaration**

Declaration of function is simply declaring the name of the function, the arguments and their types and the return type of the function. User needs to declare a function prior to the definition of the main() function when the definition of the function is written after the definition of the main() function. If the user writes the definition of the function prior to the definition of the main() function then it is not needed to declare the function explicitly.

//declaring an user defined function which has been defined later

float addNumbers(float a, float b);

Function Definition

Function definition includes the parts of the function declaration along with the body or code-block for the function. User can define a function before or after the main() function.

//defining an user defined function which has been declared earlier.

**float addNumbers(float a, float b){**

**return (a+b);**

**}**

Function Call

Calling an user defined function is similar to calling a library function, write the name of the function and provide the arguments. Also, if you need to store the returned data in a variable, then assign this call to a variable.

**int main(){**

**float result;**

**/\* calling user defined function from the main function \*/**

**result = addNumbers(.5, .8);**

**return 0;**

**}**

In this part of the example, we are calling the addNumbers() function and assigning it’s returned value to a variable result.

**Return Values and their Types**

a function can return something, otherwise does not return anything. So we can categorize them into four types.

* Function with No argument and No return type.
* Function with No argument and Return something.
* A function that takes argument but returns nothing.
* Functions that take an argument and also return something.

## Example

#include <stdio.h>

void my\_function() {

   printf("This is a function that takes no argument, and returns nothing.");

}

main() {

   my\_function();

}

## Output

This is a function that takes no argument, and returns nothing.

Here this function is not taking any input argument, and also the return type is void. So this returns nothing.

## Example

#include <stdio.h>

int my\_function() {

   printf("This function takes no argument, But returns 50  
");

   return 50;

}

main() {

   int x;

   x = my\_function();

   printf("Returned Value: %d", x);

}

## Output

This function takes no argument, But returns 50

Returned Value: 50

Here this function is not taking any input argument, but its return type is int. So this returns a value.

## Example

#include <stdio.h>

void my\_function(int x) {

   printf("This function is taking %d as argument, but returns nothing", x);

   return 50;

}

main() {

   int x;

   x = 10;

   my\_function(x);

}

## Output

This function is taking 10 as argument, but returns nothing

Here this function is taking an input argument, but its return type is void. So this returns nothing.

## Example

#include <stdio.h>

int my\_function(int x) {

   printf("This will take an argument, and will return its squared value  
");

   return x \* x;

}

main() {

   int x, res;

   x = 12;

   res = my\_function(12);

   printf("Returned Value: %d", res);

}

## Output

This function is taking 10 as argument, but returns nothing

Here this function is taking any input argument, and also returns value.

**Function Calls**

Calling a Function

While creating a C function, you give a definition of what the function has to do. To use a function, you will have to call that function to perform the defined task.

When a program calls a function, the program control is transferred to the called function. A called function performs a defined task and when its return statement is executed or when its function-ending closing brace is reached, it returns the program control back to the main program.

To call a function, you simply need to pass the required parameters along with the function name, and if the function returns a value, then you can store the returned value. For example −

[Live Demo](http://tpcg.io/T4MSFr)

#include <stdio.h>

/\* function declaration \*/

int max(int num1, int num2);

int main () {

/\* local variable definition \*/

int a = 100;

int b = 200;

int ret;

/\* calling a function to get max value \*/

ret = max(a, b);

printf( "Max value is : %d\n", ret );

return 0;

}

/\* function returning the max between two numbers \*/

int max(int num1, int num2) {

/\* local variable declaration \*/

int result;

if (num1 > num2)

result = num1;

else

result = num2;

return result;

}

## Function Declarations

A function **declaration** tells the compiler about a function name and how to call the function. The actual body of the function can be defined separately.

A function declaration has the following parts −

**return\_type function\_name( parameter list );**

For the above defined function max(), the function declaration is as follows −

**int max(int num1, int num2);**

Parameter names are not important in function declaration only their type is required, so the following is also a valid declaration −

**int max(int, int);**

Function declaration is required when you define a function in one source file and you call that function in another file. In such case, you should declare the function at the top of the file calling the function.

**Problem 9: Write a program in C to perform any 3 arithmetic operations.**

#include<stdio.h>

**int** main()

{

**int** a,b,c;

**float** x;

printf("\nEnter 2 Nos : ");

scanf("%d%d",&a,&b);

c=a+b;

printf("\nTotal : %d",c);

c=a-b;

printf("\nDifference : %d",c);

c=a\*b;

printf("\nMul : %d",c);

x=(**float**)a/(**float**)b;

printf("\nDiv : %0.2f",x);

c=a%b;

printf("\nMod : %d",c);

**return** 0;

}

To download raw file [Click Here](https://www.tutorjoes.in/c_program/08_Arithmetic.c)

#### Output

Enter 2 Nos : 25 10

Total : 35

Difference : 15

Mul : 250

Div : 2.50

Mod : 5

**Problem 10: Write a program in C to check whether the given number is prime or not.**

#include <stdio.h>

int main() {

int n, i, flag = 0;

printf("Enter a positive integer: ");

scanf("%d", &n);

// 0 and 1 are not prime numbers

// change flag to 1 for non-prime number

if (n == 0 || n == 1)

flag = 1;

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

// if n is divisible by i, then n is not prime

// change flag to 1 for non-prime number

if (n % i == 0) {

flag = 1;

break;

}

}

// flag is 0 for prime numbers

if (flag == 0)

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

else

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

return 0;

}

[Run Code](https://www.programiz.com/c-programming/online-compiler)

**Output**

Enter a positive integer: 29

29 is a prime number.