Rajarata University of Sri Lanka

Faculty of Applied Sciences

COM 1407 – Computer Programming

Practical – Functions in C

Outline

- Defining a function
- Calling a function
- Function Arguments
- Scope of a variable
- Recursion of a function

Outcome

- Be Familiar with functions in C
- Get knowledge about general structures of functions
- Write C programs using functions

A function is a block of organized, reusable code that we use to perform a single, related action. Functions provide better modularity for your application and a high degree of code reusing. You have already seen various functions like **printf()** and **main()**. These are called built-in functions provided by the language itself, but we can write our own functions as well and this tutorial will teach you how to write and use those functions in C programming language.

1 Defining a Function

- 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.
- Functions can be in built or user defined.
- 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.
- **Syntax**

```
return_type function_name( parameter list )
{
   body of the function
}
```

Simple description about the parts of a function is given below.

i. **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**.

ii. **Function Name -** This is the actual name of the function.

iii. **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.

iv. **Function Body** - The function body contains a collection of statements that define what the function does.

- Try below example for further understanding

```
#include <stdio.h>

int sum(num1,num2)
{
   int result;
   result = num1+num2;
   printf("%d",result);

   return result;

}
```

2 Function Declaration/Prototyping

- 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.
- **Syntax**

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

- Function prototype comes before the main function. The definition may appear after the main function.

- See following example.

```
#include <stdio.h>
int sub(int num1,int num2)←Function Prototype

int main () {
sub(&10,&2);
return 0;
}

int sub(int num1,int num2)←Function Definition
{
int result;
result = num1-num2;
printf("%d",result);
return result;

}
```

3 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. 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.
- Try below example for further understanding

```
#include <stdio.h>
int sum(num1,num2)← Function Definition
{
  int result;
  result = num1+num2;
  printf("%d",result);

  return result;

}
int main ()
{
  sum(1,2);←Calling the function
  return 0;
}
```

4 Function Arguments

- If a function is to use arguments, it must declare variables that accept the values of the arguments. These variables are called the **formal parameters** of the function.
- There are two ways of passing the arguments when calling a function.
  - **i.** **Call by Value** - copies the actual value of an argument into the formal parameter of the function. C programming uses call by value to pass arguments. In general, it means the code within a function cannot alter the arguments used to call the function.

```
#include <stdio.h>
int sub(num1,num2)
{
   int result;
   result = num1-num2;
   printf("%d",result);

   return result;

}
int main ()
{
   sub(10,2);
   return 0;
}
```

  - **ii.** **Call by Reference** - copies the address of an argument into the formal parameter. Inside the function, the address is used to access the actual argument used in the call. It means the changes made to the parameter affect the passed argument. You need to declare the function parameters as pointer types.

```
#include <stdio.h>

void swap(int *x, int *y)
{

   int temp;
   temp = *x;
   *x = *y;
   *y = temp;
   return;
}

int main ()
{


   int a = 100;
   int b = 200;

   printf("Before swap, value of a : %d\n", a );
   printf("Before swap, value of b : %d\n", b );


   swap(&a, &b);

   printf("After swap, value of a : %d\n", a );
   printf("After swap, value of b : %d\n", b );

   return 0;
}
```

5 Scope of a Variable

- A scope in any programming is a region of the program where a defined variable can have its existence and beyond that variable it cannot be accessed.
- There are two types of variables according to the scope.

  I. Local Variable - Variables that are declared inside a function or block are called local variables. They can be used only by statements that are inside that function or block of code.

```
#include <stdio.h>
 int main () {

   /* local variable declaration */
  int a, b;
  int c;

  /* actual initialization */
  a = 10;
  b = 20;
  c = a + b;

  printf ("value of a = %d, b = %d and c = %d\n", a, b,
 c);
  return 0;}
```

  II. **Global Variable** - Global variables are defined outside a function, usually on top of the program. Global variables hold their values throughout the lifetime of your program and they can be accessed inside any of the functions defined for the program.

```
#include <stdio.h>

/* global variable declaration */
int g;

int main () {

  int a, b;
  a = 20;
  b = 30;
  g = a + b;
  printf ("value of a = %d, b = %d and g = %d\n", a, b,
g);
  return 0;
}
```

- A program can have same name for local and global variables but the value of local variable inside a function will take preference. Try below example for further understanding.

```
#include <stdio.h>

/* global variable declaration */
int g = 20;

int main () {

 /* local variable declaration */
 int g = 10;

 printf ("value of g = %d\n",  g);
 return 0;
}
```

6 Recursion

- Recursion is the process of repeating items in a self-similar way. In programming languages, if a program allows you to call a function inside the same function, then it is called a recursive call of the function.
- **Syntax**

```
void recursion() {
 recursion(); /* function calls itself */
}

int main() {
   recursion();
}
```

- You have to define an exit condition from the function, otherwise it will go into an infinite loop.
- This recursion helps to solve many mathematical problems like factorial of a number, Fibonacci series etc.
- Try below example for further understanding.

```
#include <stdio.h>

int fibonacci(int i) {

   if(i == 0) {
      return 0;
   }

   if(i == 1) {
      return 1;
   }
   return fibonacci(i-1) + fibonacci(i-2);
}

int  main() {

   int i;

   for (i = 0; i < 10; i++) {
      printf("%d\t\n", fibonacci(i));
   }

   return 0;
}
```

7 Exercises

a. Write a C program to find square of any number using functions.(number is given by the user)

b. Write a C program to find circumference and area of a circle when user input the value of radius using functions.

c. Write a program to check whether a number is odd or even using functions.(User input the number)

d. Write a C function that takes three integers as arguments and returns the value of the largest one.

e. Write a C program for simple calculator using functions. Program will prompt the user to choose the operation choice (Add, Subtraction, Multiplication, Division, and Modulus). Then it asks the user to input two integer vales for the calculation.

f. Write a recursive function in C programming to print all natural numbers between 1 to n.

g. Write a recursive function in C programming to print all even or odd numbers between 1 to n.