Programming in C

Dr. Rupali P. Patil

Department of Electronics and Telecommunications

Module 5:
Functions

Functions

- User Defined Functions: Need, Function Declaration and Definition, Return Values, Function Calls, Passing Arguments to a Function by Value, Recursive functions, Storage classes of Variables, Command Line Arguments
- Reference: https://www.javatpoint.com/functions-in-c

Functions

- A function is a block of statements that performs a specific task. Let's say you are writing a C program and you need to perform a same task in that program more than once. In such case you have two options:

a) Use the same set of statements every time you want to perform the task
b) Create a function to perform that task, and just call it every time you need to perform that task.

- Using option (b) is a good practice and a good programmer always uses functions while writing code in C.

Functions

Why we need functions in C

Functions are used because of following reasons –

- To improve the readability of code.
- Improves the reusability of the code, same function can be used in any program rather than writing the same code from scratch.
- Debugging of the code would be easier if you use functions, as errors are easy to be traced.
- Reduces the size of the code, duplicate set of statements are replaced by function calls.

Functions

Types of functions

1) Predefined standard library functions

Standard library functions are also known as built-in functions. Functions such as puts(), gets(), printf(), scanf() etc are standard library functions. These functions are already defined in header files (files with .h extensions are called header files such as stdio.h), so we just call them whenever there is a need to use them.

For example, printf() function is defined in <stdio.h> header file so in order to use the printf() function, we need to include the <stdio.h> header file in our program using #include <stdio.h>.

Functions

Types of functions

2) User Defined functions

The functions that we create in a program are known as user defined functions or in other words you can say that a function created by user is known as user defined function.

Now we will learn how to create user defined functions and how to use them in C Programming

Functions

User Defined functions : Syntax of a function

return_type function_name (argument list)
{
    Set of statements – Block of code
}

return_type: Return type can be of any data type such as int, double, char, void, short etc.

function_name: It can be anything, however it is advised to have a meaningful name for the functions so that it would be easy to understand the purpose of function just by seeing it's name.

argument list: Argument list contains variables names along with their data types. These arguments are kind of inputs for the function. For example – A function which is used to add two integer variables, will be having two integer argument.

Block of code: Set of C statements, which will be executed whenever a call will be made to the function.

Functions

Lets take an example – **Suppose you want to create a function to add two integer variables.**

For example lets take the name addition for this function.

return_type addition(argument list)

This function addition adds two integer variables, which means we need two integer variable as input, lets provide two integer parameters in the function signature. The function signature would be –

return_type addition(int num1, int num2)

The result of the sum of two integers would be integer only. Hence function should return an integer value – we got our **return type** – It would be integer –

int addition(int num1, int num2);

So we got our function prototype or signature. Now we can implement the logic in C program like this:

Functions

Creating a void user defined function that doesn’t return anything

```
#include <stdio.h>
/* function return type is void and it doesn't have parameters*/
void introduction()
{
   printf("Hi\n");
   printf("My name is Rupali \n");
   printf("How are you?");
   /* There is no return statement inside this function, since its * return type is void     */
}

int main()
{
    /*calling function*/
    introduction();
    return 0;
}
```

Functions

Few Points to Note regarding functions in C:

1) main() in C program is also a function.

2) Each C program must have at least one function, which is main().

3) There is no limit on number of functions; A C program can have any number of functions.

4) A function can call itself and it is known as “Recursion“.

Functions

C Functions Terminologies that must remembered

return type: Data type of returned value. It can be void also, in such case function doesn't return any value.

Note: for example, if function return type is char, then function should return a value of char type and while calling this function the main() function should have a variable of char data type to store the returned value.

Functions

Task

Write function for addition of two integers, multiplication of two float numbers and display of character

Functions

Function Calls, Passing Arguments to a Function by Value

1) Function – Call by value method – In the call by value method the actual arguments are copied to the formal arguments, hence any operation performed by function on arguments doesn't affect actual parameters.

2) Function – Call by reference method – Unlike call by value, in this method, address of actual arguments (or parameters) is passed to the formal parameters, which means any operation performed on formal parameters affects the value of actual parameters.

Functions

Function – Call by value method

Actual parameters: The parameters that appear in function calls.
(e.g. num1, num2)

Formal parameters: The parameters that appear in function declarations.
(e.g.var1, var2)

What is Function Call By value?

When we pass the actual parameters while calling a function then this is known as function call by value. In this case the values of actual parameters are copied to the formal parameters. Thus operations performed on the formal parameters don't reflect in the actual parameters.

Functions

Function – Call by value method

```
#include <stdio.h>
int increment(int var)
{
    var = var+1;
    return var;
}

int main()
{
   int num1=20;
   int num2 = increment(num1);
   printf("num1 value is: %d", num1);
   printf("\nnum2 value is: %d", num2);

   return 0;
}
```

Output: ?

Functions

Function – Call by value method

```
#include <stdio.h>
int increment(int var)
{
    var = var+1;
    return var;
}

int main()
{
   int num1=20;
   int num2 = increment(num1);
   printf("num1 value is: %d", num1);
   printf("\nnum2 value is: %d", num2);

   return 0;
}
```

Output:
num1 value is: 20
num2 value is: 21

Functions

Swapping numbers using Function Call by Value

```
#include <stdio.h>
void swapnum( int var1, int var2 )
{
   int tempnum ;

   tempnum = var1 ;
   var1 = var2 ;
   var2 = tempnum ;

}
int main( )
{
    int num1 = 35, num2 = 45 ;
    printf("Before swapping: %d, %d", num1, num2);

    /*calling swap function*/
    swapnum(num1, num2);
    printf("\nAfter swapping: %d, %d", num1, num2);
}
```

Output:?

Functions

Swapping numbers using Function Call by Value

```
#include <stdio.h>
void swapnum( int var1, int var2 )
{
   int tempnum ;

   tempnum = var1 ;
   var1 = var2 ;
   var2 = tempnum ;

}
int main( )
{
    int num1 = 35, num2 = 45 ;
    printf("Before swapping: %d, %d", num1, num2);

    /*calling swap function*/
    swapnum(num1, num2);
    printf("\nAfter swapping: %d, %d", num1, num2);
}
```

Output:
Before swapping: 35, 45
After swapping: 35, 45

The reason is – function is called by value for num1 & num2. So actually var1 and var2 gets swapped (not num1 & num2). As in call by value actual parameters are just copied into the formal parameters.

Functions

Function – Call by reference method

Actual parameters: The parameters that appear in function calls.
(e.g. num1, num2)

Formal parameters: The parameters that appear in function declarations.
(e.g.var1, var2)

What is Function Call By reference?

Unlike call by value, in this method, address of actual arguments (or parameters) is passed to the formal parameters, which means any operation performed on formal parameters affects the value of actual parameters.

Functions

Swapping numbers using Function Call by reference

```
#include <stdio.h>
int swapnum ( int *var1, int *var2 )
{
  int tempnum ;
  tempnum = *var1 ;
  *var1 = *var2 ;
  *var2 = tempnum ;
}
int main( )
{
  int num1 = 35, num2 = 45 ;
  printf("Before swapping:");
  printf("\nnum1 value is %d", num1);
  printf("\nnum2 value is %d", num2);

  /*calling swap function*/
  swapnum( &num1, &num2 );

  printf("\nAfter swapping:");
  printf("\nnum1 value is %d", num1);
  printf("\nnum2 value is %d", num2);
  return 0;
}
```

Output:?

Functions

Swapping numbers using Function Call by reference

```
#include <stdio.h>
int swapnum ( int *var1, int *var2 )
{
  int tempnum ;
  tempnum = *var1 ;
  *var1 = *var2 ;
  *var2 = tempnum ;
}
int main( )
{
  int num1 = 35, num2 = 45 ;
  printf("Before swapping:");
  printf("\nnum1 value is %d", num1);
  printf("\nnum2 value is %d", num2);

  /*calling swap function*/
  swapnum( &num1, &num2 );

  printf("\nAfter swapping:");
  printf("\nnum1 value is %d", num1);
  printf("\nnum2 value is %d", num2);
  return 0;
}
```

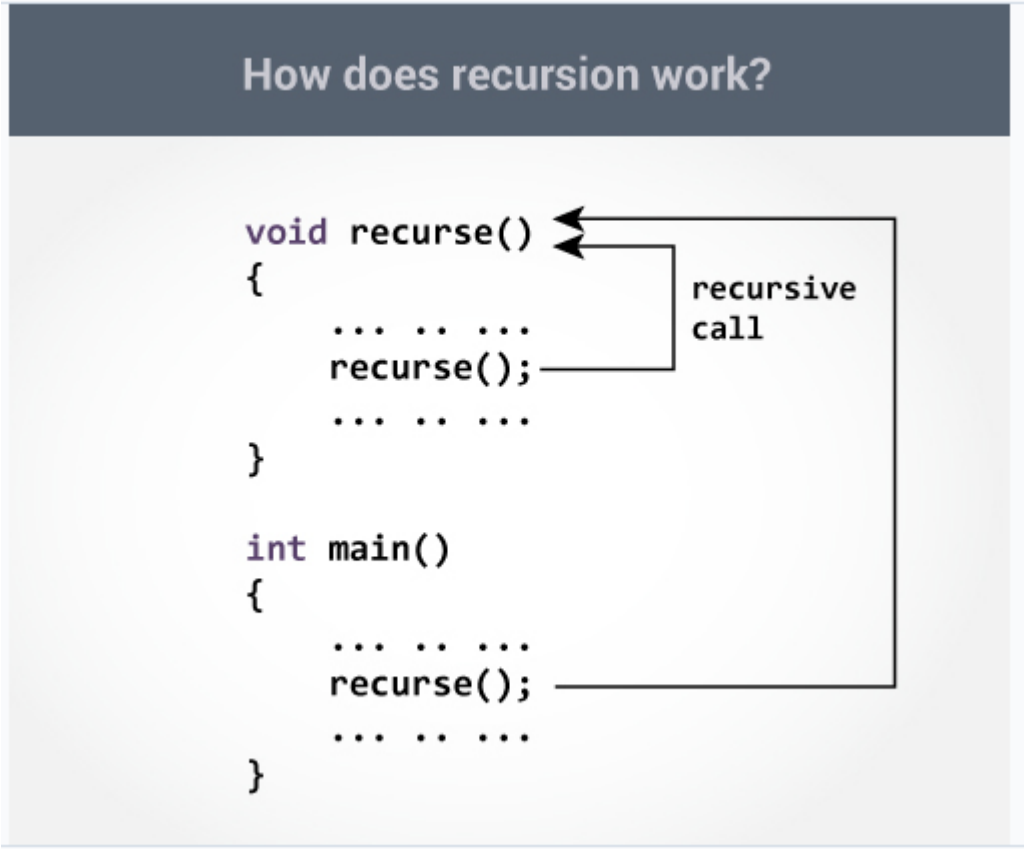
Output:
Before swapping:
num1 value is 35
num2 value is 45
After swapping:
num1 value is 45
num2 value is 35

The values of the variables have been changed after calling the swapnum() function because the swap happened on the addresses of the variables num1 and num2.

Functions

Function – Recursion.

A function that calls itself is known as a recursive function. And, this technique is known as recursion.



Functions

Function – Recursion---Example: Sum of Natural Numbers Using Recursion

```
#include <stdio.h>

int sum(int n);

int main() {
    int number, result;

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

    result = sum(number);

    printf("sum = %d", result);
    return 0;
}
```

```
int sum(int n)
{
    if (n != 0)
        // sum() function calls itself
        return n + sum(n-1);
    else
        return n;
}
```

Functions

Task

Find factorial of a number using recursion