Structured Programming using C

RCP2SFCES101

Unit-4

Functions and Parameter

Contents

Function

Introduction of Function

- A **function** is a block of code that performs a specific task
- Functions allow you to break down complex problems into smaller, manageable parts, making the code more modular, readable, and easier to maintain.
- Functions also enable code re-usability since you can define a function once and use it multiple times within a program

Types of Function

There are two types of functions in C:

1. Library Functions (Predefined Functions)
   - These are built-in functions provided by C's standard library.
   - **Examples** include printf(), scanf(), sqrt(), etc.
   - They are available for use without the need to define them and must be included using relevant header files (e.g., <stdio.h>, <string.h>).
2. User-defined Functions
   - These are functions created by the programmer to perform specific tasks.
   - They follow the same syntax and are defined according to the needs of the program.
   - This type of function is used to modularize code.

Elements of User-defined Function

Elements of User-defined Function

In order to use user-defined function we need to establish following three elements related to function:

1. Function Definition
2. Function Call
3. Function Declaration (Prototype)

Function Definition

- It Contains the actual code or statements to perform the specified task.
- Function definition includes:
  - **function name:** An identifier used to call the function
  - **return type:** Indicates what type of value (if any) the function will return. Common types include int, float, char, void (no return value).
  - **parameter list:** A comma-separated list of inputs, specifying their type and name. Parameters can be empty, in which case the function is declared as function_name(void).
  - **function body:** Contains the statements and logic to perform the specific task. enclosed in curly braces {}.

Function Definition

- **Syntax of Function Definition**

```
return_type function_name(parameter-list)
{
    function body;
    return value;              // optional, based on return type
}
```

- **Example:**

```
int add(int a, int b)
{
    int c;
    c = a + b;
    return c;
}
```

Function Call

- It Calls the function to execute its body.
- It transfers control to the function definition and may pass arguments.

- **Syntax of Function Call**

  function_name(parameter-list);

- **Example:**

  int result = add(5, 3);

Function Declaration (Prototype)

- All functions in C must be declared, before they are invoked (called).
- Function declaration also known as function prototype
- Function declaration consist of four parts:
  1. return type
  2. function name
  3. parameter list
  4. terminating semicolon

- **Syntax of Function Declaration**

```
return_type function_name(parameter-list);
```

- **Example:**

```
int add(int, int);
```

Function Example

Program to perform addition of two numbers using function

```c
#include<stdio.h>

int add(int a, int b);          //Function Prototype

int main()
{
    int x, y, result;

    printf("Enter values of x and y: ");
    scanf("%d%d",&x,&y);

    result = add(x, y);         //Function Call

    printf("Sum is: %d",result);
    return 0;
}

int add(int a, int b)           //Function Definition
{
    int c = a + b;
    return c;
}
```

Output:

```
G:\SPC\Practical Programs\addUsingFunction.exe
Enter values of x and y: 5 7
Sum is: 12
--------------------------------
```

Recursive Function

- When a function calls itself directly or indirectly then it is known as the recursion
- Recursion is a technique that breaks complex problems into small sub problems, like calculating gcd number, factorials, Fibonacci series, or performing tasks on data structures like trees.
- **Key Components of a Recursive Function:**
  Base Case: The termination condition under which the recursion stops. Without a base case, the function will keep calling itself indefinitely, leading to a stack overflow.
  Recursive Case: The part of the function where it calls itself to work on a smaller instance of the problem.

Syntax of Recursive Function

```
return_type function_name(parameters)
{
    if (base_condition)
    {
        // Base case: Stop recursion.
        return result;
    }
    else
    {
        // Recursive case: Call the function again with a smaller sub-problem.
        return function_name(modified_parameters);
    }
}
```

Example-1 Recursive Function

```c
// Program to find GCD of to integers by using recursive function

#include<stdio.h>

int gcd(int a, int b)
{
    if(b == 0)
        return a;
    else
        return gcd(b, a % b);
}
int main()
{
    int num1, num2, result;

    printf("Enter value of num1 and num2: ");
    scanf("%d%d",&num1,&num2);

    result = gcd(num1, num2);

    printf("GCD: %d", result);
    return 0;
}
```

Output:

```
G:\SPC\Practical Programs\gcd.exe
Enter value of num1 and num2: 60 36
GCD: 12
--------------------------------
```

Example-2 Recursive Function

```c
// Program to calculate factorial of number using recursive function

#include<stdio.h>

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

    printf("Enter any number: ");
    scanf("%d",&n);

    result = factorial(n);

    printf("Factorial of %d is %d", n, result);
    return 0;
}
```

Output:

```
G:\SPC\Practical Programs\factorial.exe
Enter any number: 5
Factorial of 5 is 120
--------------------------------
```

Storage Classes

Storage Classes

- In C, storage classes define the scope (visibility), lifetime, and storage location of variables.
- There are four primary storage classes in C:
  1. Automatic Storage Class (auto)
  2. External Storage Class (extern)
  3. Static Storage Class (static)
  4. Register Storage Class (register)

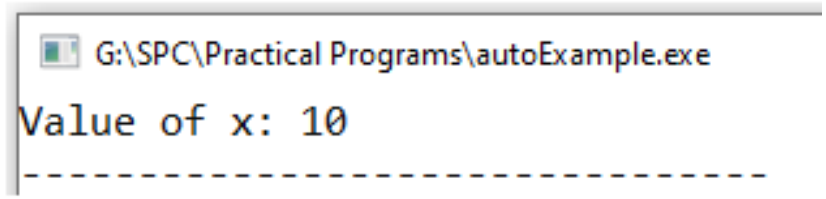
auto

- Default storage class for variables declared inside a function.
- **Scope:** Local to the block or function where the variable is declared.
- **Lifetime:** They are created when function is called and destroyed when function is exited
- **Storage:** Memory is allocated on the stack.
- **Default Initial Value:** Garbage (undefined).
- To explicitly declare automatic variable **auto** keyword is used.

Example: auto storage class

Program to demonstrate **auto** storage class

```c
#include <stdio.h>
void exampleAuto()
{
    auto int x = 10;    // 'auto' is optional, as it is the default.
    printf("Value of x: %d", x);
}
int main()
{
    exampleAuto();
    return 0;
}
```



extern

- Used for global variables to make them accessible across multiple files.
- Global variables are declared outside a function.
- **Scope:** Global (accessible from any file using extern keyword).
- **Lifetime:** Exists for the entire lifetime of the program.
- **Default Initial Value:** Zero.

Example: extern storage class

Program to demonstrate **extern** storage class

```c
#include <stdio.h>
int x = 10;                 // Global variable

void printX()
{
    extern int x;           // Declaration of external variable
    printf("Value of x: %d\n", x);
}

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

```
G:\SPC\Practical Programs\externExample.exe
Value of x: 10
```

static

- Retains its value between function calls.
- Can be applied to both local and global variables.
- **Scope:** Local if inside a function; Global if defined outside.
- **Lifetime:** Exists throughout the program's execution.
- **Default Initial Value:** Zero

Example 1: static storage class

Program to demonstrate **static** storage class

```c
#include <stdio.h>
void counter()
{
    static int count = 0;    // Retains value between calls
    count++;
    printf("Count: %d\n", count);
}

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

```
G:\SPC\Practical Programs\staticExample.exe
Count: 1
Count: 2
Count: 3
```

Example 2: static storage class

Program to demonstrate **static** storage class

```c
#include <stdio.h>

static int a = 10;  //global static variable
                    // This variable is not accessible outside this file
void display()
{
    printf("a= %d\n", a);
    a = a + 10;
}
int main()
{
    display();
    printf("a= %d", a);
    return 0;
}
```

```
G:\SPC\Practical Programs\staticExample.exe
a= 10
a= 20
--------------------------------
```

register

- **Scope:** Local to the block or function where the variable is declared.
- **Lifetime:** Exists as long as the block is executed.
- **Storage:** May be stored in a CPU register instead of RAM for faster access
- **Default Initial Value:** Garbage (undefined)

Example: register storage class

Program to demonstrate **register** storage class

```c
#include <stdio.h>
void example()
{
    register int x = 5;      // Request to store in a CPU register
    printf("Register variable x: %d", x);
}
int main()
{
    example();
    return 0;
}
```

```
G:\SPC\Practical Programs\registerExample.exe
Register variable x: 5
--------------------------------
```

References

References

- "MASTERING C" by K.R.Venugopal and Sudeep R.Prasad , Tata McGraw-Hill Publications
- "Programming in ANSI C", by E. Balaguruswamy, Tata McGraw-Hill Education.
- "Let Us C", by Yashwant Kanetkar, BPB Publication.
- "Programming in C", by Pradeep Day and Manas Gosh, Oxford University Press.
- https://www.javatpoint.com/c-programming-language-tutorial
- https://www.tutorialspoint.com/cprogramming/index.htm
- https://www.geeksforgeeks.org/c-programming-language/
- https://www.youtube.com/playlist?list=PLdo5W4Nhv31a8UcMN9-35ghv8qyFWD9_S


Thank
you