C Programming: Functions

Prof. Jyotiprakash Mishra
mail@jyotiprakash.org

January 16, 2026

Topics Covered

1. Introduction to Functions
2. Basic Functions
3. Function Prototypes
4. Call by Value
5. Return Values
6. Utility Functions
7. Array Functions
8. String Functions
9. Calculator Example
10. Variable Scope
11. Summary

What are Functions?

- Block of code that performs a specific task
- Reusable - call multiple times
- Modular - breaks program into smaller parts
- Reduces code duplication
- Easier to debug and maintain

Function Components:

- Return type - type of value returned
- Function name - identifier
- Parameters - input values (optional)
- Function body - code to execute

Why Use Functions?

- Code reusability
- Better organization
- Easier testing
- Abstraction

Function Syntax

Function Definition:

```c
return_type function_name(parameter_list) {
  // function body
  return value;  // if not void
}
```

Function Declaration (Prototype):

```c
return_type function_name(parameter_list);
```

Function Call:

```c
result = function_name(arguments);
```

Note: Prototype tells compiler about function before use

Program 1: Simple Function - No Parameters

```c
#include <stdio.h>
void greet() {
  printf("Hello, World!\n");
}
int main() {
  printf("Calling greet():\n");
  greet();
  greet();
  greet();
  printf("\nFunction called 3 times\n");
  return 0;
}
```

Output:

```
Calling greet():
Hello, World!
Hello, World!
Hello, World!

Function called 3 times
```

Explanation:

- void = no return value
- No parameters
- Called 3 times
- Reusable code block

Program 2: Function with Parameters

```c
#include <stdio.h>
void greet(char name[]) {
  printf("Hello, %s!\n", name);
}
int main() {
  greet("Alice");
  greet("Bob");
  greet("Charlie");
  return 0;
}
```

Output:

```
Hello, Alice!
Hello, Bob!
Hello, Charlie!
```

Explanation:

- Takes string parameter
- Different output each call
- Parameter = input
- Still void return

Program 3: Function with Return Value

```c
#include <stdio.h>
int square(int n) {
  return n * n;
}
int main() {
  int result;
  result = square(5);
  printf("square(5) = %d\n", result);
  result = square(10);
  printf("square(10) = %d\n", result);
  printf("square(3) = %d\n", square(3));
  return 0;
}
```

Output:

```
square(5) = 25
square(10) = 100
square(3) = 9
```

Explanation:

- Returns int value
- Takes int parameter
- Return value can be stored
- Or used directly in expression

Program 4: Multiple Parameters

```c
#include <stdio.h>
int add(int a, int b) {
  return a + b;
}
int multiply(int a, int b) {
  return a * b;
}
int main() {
  printf("add(5, 3) = %d\n",
         add(5, 3));
  printf("multiply(5, 3) = %d\n",
         multiply(5, 3));
  printf("add(10, 20) = %d\n",
         add(10, 20));
  return 0;
}
```

Output:

```
add(5, 3) = 8
multiply(5, 3) = 15
add(10, 20) = 30
```

Note:

- Multiple parameters separated by comma
- Parameter order matters
- Multiple functions in one program

Program 5: Function Prototype

```c
#include <stdio.h>
int max(int, int);
int main() {
  int a = 10, b = 20;
  printf("a = %d, b = %d\n", a, b);
  printf("Maximum: %d\n", max(a, b));
  printf("max(5, 15): %d\n", max(5, 15));
  return 0;
}
int max(int x, int y) {
  if (x > y) {
    return x;
  } else {
    return y;
  }
}
```

Output:

```
a = 10, b = 20
Maximum: 20
max(5, 15): 15
```

Explanation:

- Prototype before main
- Definition after main
- Parameter names optional in prototype
- Allows any call order

Program 6: Multiple Prototypes

```c
#include <stdio.h>
int add(int, int);
int subtract(int, int);
int multiply(int, int);
int main() {
  int a = 10, b = 5;
  printf("a = %d, b = %d\n\n", a, b);
  printf("add: %d\n", add(a, b));
  printf("subtract: %d\n",
         subtract(a, b));
  printf("multiply: %d\n",
         multiply(a, b));
  return 0;
}
int add(int x, int y) {
  return x + y;
}
int subtract(int x, int y) {
  return x - y;
}
int multiply(int x, int y) {
  return x * y;
}
```

Output:

```
a = 10, b = 5

add: 15
subtract: 5
multiply: 50
```

Note:

- All prototypes at top
- Definitions at bottom
- Clean organization
- Standard pattern

Program 7: Call by Value Demonstration

```c
#include <stdio.h>
void modify(int x) {
  x = 100;
  printf("Inside function: %d\n", x);
}
int main() {
  int num = 10;
  printf("Before call: %d\n", num);
  modify(num);
  printf("After call: %d\n", num);
  printf("\nOriginal unchanged!\n");
  return 0;
}
```

Output:

```
Before call: 10
Inside function: 100
After call: 10

Original unchanged!
```

Explanation:

- Copy of value passed
- Original not affected
- Changes only in function
- This is ”call by value”

Program 8: Swap - Call by Value Fails

```c
#include <stdio.h>
void swap(int a, int b) {
  int temp = a;
  a = b;
  b = temp;
  printf("Inside swap: a=%d, b=%d\n",
         a, b);
}
int main() {
  int x = 5, y = 10;
  printf("Before: x=%d, y=%d\n", x, y);
  swap(x, y);
  printf("After: x=%d, y=%d\n", x, y);
  printf("\nSwap didn't work!\n");
  printf("Need pointers for swap\n");
  return 0;
}
```

Output:

```
Before: x=5, y=10
Inside swap: a=10, b=5
After: x=5, y=10

Swap didn't work!
Need pointers for swap
```

Note:

- Values swapped in function
- Original variables unchanged
- Call by value limitation
- Pointers needed for swap

Program 9: Multiple Return Statements

```c
#include <stdio.h>
int absolute(int n) {
  if (n < 0) {
    return -n;
  } else {
    return n;
  }
}
int main() {
  printf("absolute(-5) = %d\n",
         absolute(-5));
  printf("absolute(10) = %d\n",
         absolute(10));
  printf("absolute(0) = %d\n",
         absolute(0));
  return 0;
}
```

Output:

```
absolute(-5) = 5
absolute(10) = 10
absolute(0) = 0
```

Note:

- Multiple return paths
- Only one executes
- Returns immediately
- Function ends at return

Program 10: Return from Anywhere

```c
#include <stdio.h>
int findFirst(int arr[], int size,
              int target) {
  int i;
  for (i = 0; i < size; i++) {
    if (arr[i] == target) {
      return i;
    }
  }
  return -1;
}
int main() {
  int arr[] = {10, 20, 30, 40, 50};
  printf("Array: 10 20 30 40 50\n\n");
  printf("Find 30: index %d\n",
         findFirst(arr, 5, 30));
  printf("Find 99: index %d\n",
         findFirst(arr, 5, 99));
  printf("\n-1 means not found\n");
  return 0;
}
```

Output:

```
Array: 10 20 30 40 50

Find 30: index 2
Find 99: index -1

-1 means not found
```

Logic:

- Return from loop when found
- Return -1 if not found
- Early exit optimization

Program 11: Factorial Function

```c
#include <stdio.h>
int factorial(int n) {
  int result = 1;
  int i;
  for (i = 1; i <= n; i++) {
    result *= i;
  }
  return result;
}
int main() {
  int i;
  printf("Factorials:\n");
  for (i = 0; i <= 6; i++) {
    printf("%d! = %d\n",
           i, factorial(i));
  }
  return 0;
}
```

Output:

```
Factorials:
0! = 1
1! = 1
2! = 2
3! = 6
4! = 24
5! = 120
6! = 720
```

Note:

- Iterative (not recursive)
- Uses loop
- Returns calculated value

Program 12: Prime Number Check

```c
#include <stdio.h>
int isPrime(int n) {
  int i;
  if (n <= 1) return 0;
  for (i = 2; i * i <= n; i++) {
    if (n % i == 0) {
      return 0;
    }
  }
  return 1;
}
int main() {
  int i;
  printf("Prime numbers 1-20:\n");
  for (i = 1; i <= 20; i++) {
    if (isPrime(i)) {
      printf("%d ", i);
    }
  }
  printf("\n");
  return 0;
}
```

Output:

```
Prime numbers 1-20:
2 3 5 7 11 13 17 19
```

Logic:

- Returns 1 if prime
- Returns 0 if not prime
- Boolean-like function
- Check up to square root

Program 13: Power Function

```c
#include <stdio.h>
int power(int base, int exp) {
  int result = 1;
  int i;
  for (i = 0; i < exp; i++) {
    result *= base;
  }
  return result;
}
int main() {
  printf("2^3 = %d\n", power(2, 3));
  printf("5^2 = %d\n", power(5, 2));
  printf("3^4 = %d\n", power(3, 4));
  printf("10^0 = %d\n", power(10, 0));
  return 0;
}
```

Output:

```
2^3 = 8
5^2 = 25
3^4 = 81
10^0 = 1
```

Note:

- base raised to exp
- Iterative approach
- Works for non-negative exp
- Returns 1 for exp $= 0$

Program 14: GCD Function

```c
#include <stdio.h>
int gcd(int a, int b) {
  int temp;
  while (b != 0) {
    temp = b;
    b = a % b;
    a = temp;
  }
  return a;
}
int main() {
  printf("gcd(48, 18) = %d\n",
         gcd(48, 18));
  printf("gcd(100, 50) = %d\n",
         gcd(100, 50));
  printf("gcd(17, 13) = %d\n",
         gcd(17, 13));
  return 0;
}
```

Output:

```
gcd(48, 18) = 6
gcd(100, 50) = 50
gcd(17, 13) = 1
```

Note:

- Euclidean algorithm
- Uses while loop
- Returns greatest common divisor
- GCD of coprime = 1

Program 15: Array Sum Function

```c
#include <stdio.h>
int arraySum(int arr[], int size) {
  int sum = 0;
  int i;
  for (i = 0; i < size; i++) {
    sum += arr[i];
  }
  return sum;
}
int main() {
  int nums[] = {10, 20, 30, 40, 50};
  int size = 5;
  printf("Array: ");
  for (int i = 0; i < size; i++) {
    printf("%d ", nums[i]);
  }
  printf("\n\nSum: %d\n",
         arraySum(nums, size));
  return 0;
}
```

Output:

```
Array: 10 20 30 40 50

Sum: 150
```

Note:

- Array passed as parameter
- Must also pass size
- Array name = pointer
- Size not known inside function

Program 16: Find Maximum in Array

```c
#include <stdio.h>
int findMax(int arr[], int size) {
  int max = arr[0];
  int i;
  for (i = 1; i < size; i++) {
    if (arr[i] > max) {
      max = arr[i];
    }
  }
  return max;
}
int main() {
  int nums[] = {34, 12, 89, 5, 67};
  printf("Array: 34 12 89 5 67\n");
  printf("Maximum: %d\n",
         findMax(nums, 5));
  return 0;
}
```

Output:

```
Array: 34 12 89 5 67
Maximum: 89
```

Logic:

- Assume first is max
- Compare with rest
- Update if larger found
- Return maximum

Program 17: Print Array Function

```c
#include <stdio.h>
void printArray(int arr[], int size) {
  int i;
  printf("[ ");
  for (i = 0; i < size; i++) {
    printf("%d", arr[i]);
    if (i < size - 1) {
      printf(", ");
    }
  }
  printf(" ]\n");
}
int main() {
  int arr1[] = {1, 2, 3, 4, 5};
  int arr2[] = {10, 20, 30};
  printf("Array 1: ");
  printArray(arr1, 5);
  printf("Array 2: ");
  printArray(arr2, 3);
  return 0;
}
```

Output:

```
Array 1: [ 1, 2, 3, 4, 5 ]
Array 2: [ 10, 20, 30 ]
```

Note:

- Void function
- Just prints, no return
- Reusable display
- Nice formatting

Program 18: String Length Function

```c
#include <stdio.h>
int stringLength(char str[]) {
  int len = 0;
  while (str[len] != '\0') {
    len++;
  }
  return len;
}
int main() {
  char str1[] = "Hello";
  char str2[] = "Programming";
  printf("str1: \"%s\"\n", str1);
  printf("Length: %d\n\n",
         stringLength(str1));
  printf("str2: \"%s\"\n", str2);
  printf("Length: %d\n",
         stringLength(str2));
  return 0;
}
```

Output:

```
str1: "Hello"
Length: 5

str2: "Programming"
Length: 11
```

Note:

- Manual strlen
- Count until \0
- String = char array
- Returns int length

Program 19: String Copy Function

```c
#include <stdio.h>
void stringCopy(char dest[], char src[]) {
  int i = 0;
  while (src[i] != '\0') {
    dest[i] = src[i];
    i++;
  }
  dest[i] = '\0';
}
int main() {
  char src[] = "Hello";
  char dest[20];
  printf("Source: %s\n", src);
  stringCopy(dest, src);
  printf("Destination: %s\n", dest);
  return 0;
}
```

Output:

```
Source: Hello
Destination: Hello
```

Note:

- Manual strcpy
- Copy char by char
- Add null terminator
- Void function (modifies dest)

Program 20: Calculator with Functions

```c
#include <stdio.h>
int add(int a, int b) { return a+b; }
int sub(int a, int b) { return a-b; }
int mul(int a, int b) { return a*b; }
int div(int a, int b) { return a/b; }
void showMenu() {
  printf("\nCalculator\n");
  printf("1. Add\n2. Subtract\n");
  printf("3. Multiply\n4. Divide\n");
}
int main() {
  int a = 10, b = 5, choice = 1;
  showMenu();
  printf("\nChoice: %d\n", choice);
  printf("Numbers: %d, %d\n\n", a, b);
  if (choice == 1)
    printf("Result: %d\n", add(a,b));
  else if (choice == 2)
    printf("Result: %d\n", sub(a,b));
  else if (choice == 3)
    printf("Result: %d\n", mul(a,b));
  else if (choice == 4)
    printf("Result: %d\n", div(a,b));
  return 0;
}
```

Output:

```
Calculator
1. Add
2. Subtract
3. Multiply
4. Divide

Choice: 1
Numbers: 10, 5

Result: 15
```

Note:

- Multiple small functions
- Each does one thing
- Modular design
- Easy to maintain

Program 21: Local vs Global Variables

```c
#include <stdio.h>
int global = 100;
void testScope() {
  int local = 50;
  printf("Inside function:\n");
  printf("  local = %d\n", local);
  printf("  global = %d\n", global);
  global = 200;
}
int main() {
  printf("Before call:\n");
  printf("  global = %d\n\n", global);
  testScope();
  printf("\nAfter call:\n");
  printf("  global = %d\n", global);
  return 0;
}
```

Output:

```
Before call:
  global = 100

Inside function:
  local = 50
  global = 100

After call:
  global = 200
```

Note:

- Global: accessible everywhere
- Local: only in function
- Global can be modified
- Local destroyed after function

Functions - Summary

Key Points:

- Function = reusable code block
- Prototype declares, definition implements
- Parameters = input, return = output
- Call by value = copy passed
- void = no return value
- Multiple returns possible
- Arrays passed with size
- Local variables in function scope
- Global variables accessible everywhere
- Break programs into functions

Function Components

Component	**Description**
Return type	Type of value returned (void if none)
Function name	Identifier for the function
Parameters	Input values (optional)
Function body	Code to execute
Return statement	Returns value (if not void)

Example:

- `int add(int a, int b) { return a+b; }`
- Return type: `int`
- Name: `add`
- Parameters: `int a, int b`
- Body: `{ return a+b; }`

Best Practices

1. **One task per function** - single responsibility
2. **Use prototypes** for better organization
3. **Meaningful names** - describe what it does
4. **Keep functions short** - easier to understand
5. **Document parameters** - comment what they mean
6. **Validate input** - check parameters
7. **Use const** for parameters that shouldn't change
8. **Return error codes** - use -1, NULL for errors
9. **Minimize global variables** - use parameters
10. **Test functions** independently

Common Mistakes

1. **Missing prototype** - declaration before use
2. **Type mismatch** - return type vs actual return
3. **Missing return** - non-void function must return
4. **Wrong parameter count** - must match definition
5. **Parameter order** - position matters
6. **Modifying local copy** - call by value limitation
7. **Array size unknown** - must pass size separately
8. **Returning local address** - undefined behavior
9. **Infinite loops** - in iterative functions
10. **Not initializing** - local variables

Practice Exercises

Try these programs:

1. Write fibonacci function (iterative)
2. Check if number is perfect square
3. Reverse an array using function
4. Find LCM of two numbers
5. Count digits in a number
6. Convert decimal to binary (iterative)
7. Check if string is palindrome
8. Bubble sort function
9. Linear search function
10. Matrix addition function