PROGRAMMING IN C

Chapter 4: Functions

Exam-Oriented Questions with Solutions

This question set is arranged in the pattern usually followed in semester/board examinations — very short answer questions (1 mark), short answer questions (3 marks), long/descriptive answer questions (5 marks), and output-prediction questions. Model answers are given for every question so that students can check their preparation and also use them directly for quick revision before the exam.

Group A — Very Short Answer Type Questions (1 Mark each)

Answer each of the following in one sentence.

Q1. Define a function. *[1 Mark]*

Answer:

A function is a self-contained block of program statements that performs a specific, well-defined task and can be called (invoked) from other parts of the program.

Q2. What is a function prototype? *[1 Mark]*

Answer:

A function prototype is a declaration that specifies a function's name, return type, and the type of its parameters, before the function is defined or used, so that the compiler can check calls to it.

Q3. What is the default storage class of a local variable declared inside a function? *[1 Mark]*

Answer:

The default storage class of a local variable is auto (automatic).

Q4. Name the two methods of passing arguments to a function in C. *[1 Mark]*

Answer:

Call by value and call by reference.

Q5. What is recursion? *[1 Mark]*

Answer:

Recursion is a technique in which a function calls itself, directly or indirectly, to solve a problem by breaking it into smaller sub-problems of the same type.

Q6. What is meant by the scope of a variable? *[1 Mark]*

Answer:

Scope is the region or part of the program within which a variable can be accessed or referred to by its name.

Q7. Which storage class specifier makes a variable retain its value between successive function calls? *[1 Mark]*

Answer:

The static storage class.

Q8. What is the default initial value of an uninitialized static variable? *[1 Mark]*

Answer:

Zero.

Q9. Write the general syntax of a function prototype. *[1 Mark]*

Answer:

return_type function_name(parameter_type_1, parameter_type_2, ...);

Q10. What is the purpose of the return statement in a function? *[1 Mark]*

Answer:

The return statement ends the execution of a function and sends a value (if any) back to the point from where the function was called.

Group B — Short Answer Type Questions (3 Marks each)

Answer each of the following in about 3–5 sentences or points.

Q1. Differentiate between call by value and call by reference. *[3 Marks]*

Answer:

Basis	Call by Value	Call by Reference
What is passed	A copy of the actual value	The address of the actual variable
Effect on original variable	No change to the original variable	Original variable is modified
Memory	Separate memory for formal parameter	Formal parameter shares the same memory (via pointer)
Default in C	Yes, this is the default method	Achieved only by explicitly using pointers

Q2. State any three reasons for the need of functions in C. *[3 Marks]*

Answer:

- Modularity: a large program can be divided into smaller, manageable, logically independent units.
- Reusability: a function, once written and tested, can be called repeatedly without rewriting the code.
- Easier debugging: since each function performs a single task, errors can be isolated and corrected within that function alone.

Q3. Differentiate between actual parameters and formal parameters. *[3 Marks]*

Answer:

Actual parameters are the values or expressions written inside the parentheses of a function call statement in the calling function, e.g., the x and y in add(x, y). Formal parameters are the variable names listed in the header of the function definition that receive these values, e.g., the a and b in int add(int a, int b). The number, order and type of actual parameters must match the formal parameters of the function being called.

Q4. Explain the terms 'scope' and 'lifetime' of a variable with a suitable example. *[3 Marks]*

Answer:

Scope refers to the part of the program in which a variable name can be legally used, while lifetime refers to the duration for which the variable remains allocated in memory. For example, a variable declared inside a function using auto has local scope, meaning it can be used only inside that function, and an automatic lifetime, meaning it is created when the function starts and destroyed when the function ends. A static local variable, in contrast, still has local scope, but its lifetime extends across the entire program run, since its value is preserved between function calls.

Q5. What are storage classes in C? Name all the storage classes available. *[3 Marks]*

Answer:

A storage class in C determines three properties of a variable — its scope, its lifetime, and its default initial value. C provides four storage classes:

- auto — the default storage class for local variables.
- register — requests storage in a CPU register for faster access.
- static — retains its value between function calls; lifetime spans the whole program.
- extern — refers to a global variable or function defined elsewhere, making it accessible across files.

Q6. Differentiate between local and global variables. *[3 Marks]*

Answer:

Basis	Local Variable	Global Variable
Place of declaration	Inside a function or block	Outside all functions
Scope	Restricted to that function/block	Accessible throughout the file
Default lifetime	Automatic (exists only during the call)	Entire program run

Q7. What is meant by the 'base case' in recursion? Why is it necessary? *[3 Marks]*

Answer:

The base case is the terminating condition in a recursive function, under which the function stops calling itself and returns a value directly instead of making a further recursive call. It is necessary because, without a base case, the function would keep calling itself indefinitely, continuously pushing new stack frames onto the call stack until the stack memory is exhausted, causing a runtime error known as stack overflow. The base case ensures that recursion eventually terminates and the pending calls can start returning.

Q8. Differentiate between direct recursion and indirect recursion. *[3 Marks]*

Answer:

In direct recursion, a function calls itself directly from within its own body, e.g., function fact() calling fact() again in its definition. In indirect (mutual) recursion, a function A calls a different function B, and function B in turn calls function A back, so that the two functions call each other in a cycle rather than a single function calling only itself.

Group C — Long Answer / Descriptive Type Questions (5 Marks each)

Answer each of the following in detail, with examples/programs wherever necessary.

Q1. Explain call by value and call by reference with a suitable C program for each. *[5 Marks]*

Answer:

Call by value: A copy of the actual parameter's value is passed to the formal parameter. Changes made inside the function affect only the local copy and not the original variable.

```
#include <stdio.h>
void modify(int x)
{
    x = x + 10;
}
int main(void)
{
    int a = 5;
    modify(a);
    printf("%d", a);   /* prints 5 */
    return 0;
}
```

Call by reference: The address of the actual variable is passed, using a pointer parameter, so the function can access and modify the original variable directly.

```
#include <stdio.h>
void modify(int *x)
{
    *x = *x + 10;
}
int main(void)
{
    int a = 5;
    modify(&a);
    printf("%d", a);   /* prints 15 */
    return 0;
}
```

Thus, call by value protects the original data from modification, while call by reference is used whenever a function needs to change the caller's variables directly, or needs to return more than one value.

Q2. Explain the storage classes available in C with a suitable example for each. *[5 Marks]*

Answer:

C provides four storage class specifiers — auto, register, static and extern — which together determine a variable's scope, lifetime, and default value, as summarised below.

Storage Class	Scope	Lifetime	Default Value
auto	Local	Within the block	Garbage
register	Local	Within the block	Garbage
static	Local (or file, if global)	Entire program	Zero
extern	Global	Entire program	Zero

Example of static, showing that its value is preserved between calls:

```
#include <stdio.h>
void counter(void)
{
    static int count = 0;
    count++;
```

```
    printf("%d\n", count);
}
int main(void)
{
    counter();  counter();  counter();
    return 0;
}
/* Output: 1  2  3 */
```

Q3. What is recursion? Explain, with the factorial function as an example, how a recursive function executes using the memory (call) stack. *[5 Marks]*

Answer:

Recursion is a technique in which a function calls itself, either directly or indirectly, to solve a problem by reducing it to smaller sub-problems of the same kind. Every correct recursive function must have a base case, which stops further recursive calls, and a recursive case, which calls the function again with a modified (usually smaller) argument.

```
int factorial(int n)
{
    if (n == 0)              /* base case */
        return 1;
    else
        return n * factorial(n - 1);   /* recursive case */
}
```

When factorial(4) is called, the compiler creates a new stack frame (activation record) containing n = 4 and the return address, and pushes it onto the call stack. Since n is not 0, the function calls factorial(3), which pushes another frame on top, and so on until factorial(0) is called. At this point the base case is satisfied and 1 is returned immediately without any further call. The calls now start returning in reverse order: factorial(1) returns 1 * 1 = 1, factorial(2) returns 2 * 1 = 2, factorial(3) returns 3 * 2 = 6, and finally factorial(4) returns 4 * 6 = 24. Each time a call returns, its stack frame is popped off the stack. If the base case were missing, calls would keep pushing frames indefinitely, eventually exhausting the stack and causing a stack overflow error.

Q4. Explain the categories of functions in C based on arguments and return values, giving an example program for each category. *[5 Marks]*

Answer:

Based on whether a function accepts arguments and whether it returns a value, functions can be classified into the following categories:

```
/* 1. No argument, no return value */
void greet(void) { printf("Hello!"); }

/* 2. No argument, with return value */
int getValue(void) { int x; scanf("%d", &x); return x; }

/* 3. Argument, with return value */
int square(int n) { return n * n; }
```

In the first category, the function neither receives data from, nor sends data back to, the caller; it usually performs an independent task such as displaying a message. In the second category, the function computes or reads a value on its own and returns it, without needing any input from the caller. In the third, and most flexible, category, the function receives one or more values as arguments, processes them, and returns a result, so that data flows in both directions between the caller and the function.

Q5. Write a C program using a static variable to generate the first n terms of a running total each time a function is called, and explain why a static variable, rather than an ordinary local variable, must be used. *[5 Marks]*

Answer:

```
#include <stdio.h>
void runningTotal(int value)
{
    static int total = 0;   /* must be static */
    total += value;
    printf("Running total = %d\n", total);
}
int main(void)
{
    runningTotal(10);
    runningTotal(20);
    runningTotal(5);
    return 0;
}
/* Output:
   Running total = 10
   Running total = 30
   Running total = 35 */
```

The variable total must be declared static because it needs to remember the accumulated sum from previous calls to runningTotal(). If total were an ordinary (automatic) local variable, it would be re-created and reset to 0 on every call, so each call would only display the value passed in that call instead of the correct running total. Because total is static, it is initialized to 0 only once, on the first call, and its value persists in memory across all subsequent calls for the rest of the program's execution.

Q6. What is a function prototype? Explain its importance and write a complete C program that declares a function prototype separately from its definition. *[5 Marks]*

Answer:

A function prototype is a declaration statement that informs the compiler about a function's name, return type, and the number and types of its parameters, before the function is actually defined or called in the program. Its general syntax is:

```
return_type function_name(parameter_type_1, parameter_type_2, ...);
```

A prototype is important because it allows the compiler to check, at the point of every function call, whether the correct number and type of arguments have been supplied, and it allows the function to be called from parts of the program that appear before its actual definition, or even from a different source file, provided the prototype is visible. Without a prototype, mismatched calls may go undetected and can lead to incorrect program behaviour.

```
#include <stdio.h>

int add(int, int);          /* function prototype */

int main(void)
{
    printf("Sum = %d", add(7, 5));   /* function used before its definition */
    return 0;
}

int add(int a, int b)       /* function definition */
{
```

```
    return a + b;
}
```

Group D — Predict the Output (with Explanation)

Carefully trace each program and write down its exact output before checking the answer given below it. Such questions are frequently asked in examinations to test the understanding of function execution.

Q1. Predict the output of the following program: *[3 Marks]*

Answer:

```
#include <stdio.h>
void show(void)
{
    static int count = 0;
    count++;
    printf("%d ", count);
}
int main(void)
{
    show();
    show();
    show();
    return 0;
}
```

Output: 1 2 3

Since count is static, it is initialized to 0 only once and retains its value across the three separate calls to show(), so it increases by 1 on each call instead of resetting.

Q2. Predict the output of the following program: *[3 Marks]*

Answer:

```
#include <stdio.h>
void change(int a)
{
    a = a * 2;
}
int main(void)
{
    int x = 10;
    change(x);
    printf("%d", x);
    return 0;
}
```

Output: 10

change(x) is a call by value, so the parameter a receives only a copy of x. Doubling a inside the function has no effect on x in main(), which remains 10.

Q3. Predict the output of the following recursive program: *[3 Marks]*

Answer:

```
#include <stdio.h>
void countdown(int n)
{
    if (n == 0)
        return;
    printf("%d ", n);
    countdown(n - 1);
}
int main(void)
```

```
{
    countdown(4);
    return 0;
}
```

Output: 4 3 2 1

countdown(4) prints 4 and calls countdown(3); countdown(3) prints 3 and calls countdown(2); this continues until countdown(0), which satisfies the base case (n == 0) and simply returns without printing anything further.

Q4. Predict the output of the following program: *[3 Marks]*

Answer:

```
#include <stdio.h>
int x = 50;
void display(void)
{
    int x = 5;
    printf("%d ", x);
}
int main(void)
{
    display();
    printf("%d", x);
    return 0;
}
```

Output: 5 50

Inside display(), the local variable x (value 5) shadows the global x for the duration of that function, so its printf() prints 5. Inside main(), there is no local x, so its printf() refers to the global x, which still holds 50.

Q5. Predict the output of the following program: *[3 Marks]*

Answer:

```
#include <stdio.h>
int square(int n)
{
    return n * n;
}
int main(void)
{
    int result = square(3) + square(2);
    printf("%d", result);
    return 0;
}
```

Output: 13

square(3) returns 9 and square(2) returns 4; these two returned values are added together in main() to give result = 9 + 4 = 13, which is then printed.

Q6. Predict the output of the following program: *[3 Marks]*

Answer:

```
#include <stdio.h>
void increment(int *p)
{
    (*p)++;
}
int main(void)
{
```

```
    int a = 7;
    increment(&a);
    increment(&a);
    printf("%d", a);
    return 0;
}
```

Output: 9

increment() receives the address of a and increases the value at that address by 1 each time it is called (call by reference). Since it is called twice, a is increased from 7 to 8 and then from 8 to 9, so the final printed value is 9.