

**Question bank Pattern**

**[subject code] [ subject name]**

**Unit 3 [FUNCTIONS ,STORAGE CLASSES & ARRAYS]**

**PART A (10x2)**

Question 1: What are the two main types of functions in the C language?

Answer: The two main types of functions in C are Library Functions and User-defined Functions.

Question 2: Explain the rules for calling a specific function in C.

Answer: To call a specific function in C, you must provide the function's name followed by parentheses and the appropriate arguments based on the function's parameter list.

Question 3: Name and briefly describe the four types of user-defined functions in C.

Answer: The four types of user-defined functions in C are:

* No return value, No argument: Functions with no return value and no arguments.
* No return value, With arguments: Functions with no return value but with arguments.
* With return value, No argument: Functions with a return value but no arguments.
* With return value, With arguments: Functions with both a return value and arguments.

Question 4: What is the difference between function call by value and function call by address (or call by reference) in C?

Answer: In a function call by value, a copy of the argument's value is passed to the function. In a function call by address, the address of the argument is passed, allowing the function to modify the original variable.

Question 5: Explain the scope of local variables in a C function.

Answer: Local variables in a C function are declared within the function's block and are only accessible within that function. They have a limited scope and are not visible to other functions or code outside the function.

Question 6: Name the four storage classes in C.

Answer: The four storage classes in C are: auto, register, static, and extern.

Question 7: How are command-line arguments passed to a C program?

Answer: Command-line arguments in C are passed to the main function as two parameters: int argc (argument count) and char\* argv[] (argument vector).

Question 8: How do you declare and initialize an array in C?

Answer: An array in C is declared and initialized as follows:

int myArray[5] = {1, 2, 3, 4, 5};

Question 9: What is the difference between the length and size of an array in C?

Answer: The length of an array in C is the number of elements it contains, while the size of an array is the total memory occupied by the array, often measured in bytes.

Question 10: How do you declare and initialize a 2D array in C?

Answer: A 2D array in C is declared and initialized like this:

int my2DArray[3][4] = {{1, 2, 3, 4}, {5, 6, 7, 8}, {9, 10, 11, 12}};

**PART B (5x13)**

Question 1 : Explain the difference between library functions and user-defined functions inC.Provide examples of each type and discuss their significance in programming.

Answer 1: Library functions in C are pre-defined functions provided by the C standard library or other libraries that can be used in a program. These functions serve common tasks such as input/output, string manipulation, and mathematical operations. Examples include printf(), scanf(), and strlen(). Library functions save time and effort in programming by providing ready-made solutions.

User-defined functions, on the other hand, are functions created by the programmer. These functions are defined to perform specific tasks or operations tailored to the program's requirements. For example, a user-defined function calculateAverage can be created to calculate the average of a set of numbers. User-defined functions enhance code modularity, readability, and reusability.

Question 2 : Discuss the rules and steps involved in calling a specific function in C. Explain how arguments are passed to functions and how the return value is received.

Answer 2 : To call a specific function in C, you need to follow these rules and steps:

1. Function Declaration: Ensure that the function has been declared or defined before its use in the program.
2. Function Name: Specify the name of the function you want to call.
3. Arguments: Provide the required arguments within parentheses if the function expects any.
4. Function Call: Call the function by using its name followed by parentheses. Pass the arguments as per the function's parameter list.
5. Return Value: If the function returns a value, capture it using a variable of the appropriate data type. This is done by assigning the result of the function call to a variable.

Example:

int result = addNumbers(5, 3); // Calling the function addNumbers with arguments

In this example, the function addNumbers is called with two arguments, and the return value is stored in the variable result.

Question 3 : Explain the concept of storage classes in C. Discuss the characteristics and use cases of the auto, static, register, and extern storage classes.

Answer 3 : Storage classes in C determine the lifetime, scope, and visibility of variables. There are four primary storage classes:

1. auto: Variables declared as auto have a local scope and a default storage duration of automatic storage. They are created when the block containing their declaration is entered and destroyed when the block is exited. This is the default storage class for local variables.
2. static: Variables declared as static have a local scope but a static storage duration. They are created only once, and their value persists across function calls. These variables are initialized to zero by default if not explicitly initialized.
3. register: Variables declared as register are used to suggest that a variable should be stored in a CPU register for faster access. However, it is up to the compiler to decide whether to honor this suggestion.
4. extern: Variables declared as extern have global scope but are defined in another file. They are typically used to access global variables defined in other files.

Use cases for each storage class:

* auto: Use for temporary local variables with a short lifespan.
* static: Use for variables that need to retain their values across function calls.
* register: Use to hint the compiler to store a variable in a register for speed optimization (not always guaranteed).
* extern: Use to access global variables defined in other source files.

Question 4 : Explain the concept of command-line arguments in C. Discuss how command-line arguments are passed to a C program and how they can be accessed within the program.

Answer 4 : Command-line arguments in C allow you to pass values to a program when it is executed. They are passed to the main function as parameters. Here's how they work:

1. Parameter List of main: The main function in C can have two parameters: int argc (argument count) and char\* argv[] (argument vector).
2. argc Parameter: argc contains the number of command-line arguments passed to the program, including the program name itself.
3. argv Parameter: argv is an array of strings where each element represents a command-line argument. argv[0] typically contains the program name, and subsequent elements hold the arguments.

Example:

int main(int argc, char\* argv[]) {

printf("Total arguments: %d\n", argc);

for (int i = 0; i < argc; i++) {

printf("Argument %d: %s\n", i, argv[i]);

}

return 0;

}

In this example, running the program as ./myprogram arg1 arg2 would produce:

Total arguments: 3

Argument 0: ./myprogram

Argument 1: arg1

Argument 2: arg2

Question 5 : Discuss arrays in C, including their declaration, initialization, accessing individual elements, finding the length, and determining the size of an array. Provide examples for each concept.

Answer 5 : Arrays in C are collections of elements of the same data type. Here are the concepts related to arrays:

1. Declaration and Initialization:
   * Declaration: int myArray[5]; declares an integer array with 5 elements.
   * Initialization: int myArray[] = {1, 2, 3, 4, 5}; initializes an array with values.
2. Accessing Individual Elements:
   * To access the i-th element: element = myArray[i];
3. Finding the Length:
   * The length of an array can be calculated using sizeof(myArray) / sizeof(myArray[0]).
4. Determining the Size:
   * The size of an array in bytes can be found using sizeof(myArray).

Example:

int myArray[5] = {1, 2, 3, 4, 5};

int element = myArray[2]; // Accessing the third element

int length = sizeof(myArray) / sizeof(myArray[0]); // Finding the length

int size = sizeof(myArray); // Determining the size