C Programming Interview Questions with Answers

Basic Level Questions

1. What is C programming language?

Answer: C is a general-purpose, procedural programming language developed by Dennis Ritchie at Bell Labs between 1971-1973. It's a middle-level language that combines features of both high-level and low-level languages, making it suitable for system programming, embedded systems, and application development.

2. Why is C called a mid-level programming language?

Answer: C is called a mid-level language because it combines features of both high-level and low-level languages. It supports high-level constructs like loops, functions, and data structures, while also allowing low-level operations like direct memory manipulation and bit-level operations using pointers.

3. What are the key features of C programming language?

Answer: Key features include:

- Simple and efficient
- Portable across platforms
- Structured programming approach
- Rich set of built-in functions
- Dynamic memory allocation
- Pointers for memory management
- Recursion support
- Extensive library functions

4. What are the basic data types in C?

Answer: The basic data types are:

- **int**: Integer values (typically 4 bytes)
- **char**: Single character (1 byte)
- **float**: Single-precision floating point (4 bytes)

- **double**: Double-precision floating point (8 bytes)
- **void**: Represents no value

5. What are tokens in C?

Answer: Tokens are the smallest individual units of a C program. They include:

- **Keywords**: Reserved words (int, char, if, else, etc.)
- **Identifiers**: User-defined names for variables, functions
- **Constants**: Fixed values (10, 'A', 3.14)
- **Strings**: Sequence of characters ("Hello")
- **Special Symbols**: Operators and punctuators (+, -, {, }, etc.)
- **Operators**: Symbols that perform operations

6. What is the difference between declaration and definition?

Answer:

- **Declaration**: Tells the compiler about the name and type of a variable/function without allocating memory
- **Definition**: Actually creates the variable/function and allocates memory

```
extern int x;     // Declaration
int x = 10;       // Definition
```

7. What are variables and constants?

Answer:

- **Variables**: Named memory locations that can store data and can be modified during program execution
- **Constants**: Fixed values that cannot be changed during program execution

```
int age = 25;          // Variable
const int MAX = 100;    // Constant
```

8. What is the scope of a variable?

Answer: Scope defines the region of the program where a variable can be accessed:

- **Local scope**: Variables declared inside a function/block
- **Global scope**: Variables declared outside all functions
- **Function scope**: Variables accessible within a function

9. What are storage classes in C?

Answer: Storage classes define the scope, visibility, and lifetime of variables:

- **auto**: Default for local variables
- **register**: Stored in CPU registers for faster access
- **static**: Retains value between function calls
- **extern**: Declares a variable defined elsewhere

10. What is the difference between auto and static variables?

Answer:

- **auto**: Default storage class, destroyed when function exits, initialized with garbage values
- **static**: Retains value between function calls, initialized only once with 0 by default

11. What are preprocessor directives?

Answer: Preprocessor directives are commands processed before compilation:

- **#include**: Includes header files
- **#define**: Defines macros
- **#ifdef, #ifndef**: Conditional compilation
- **#pragma**: Compiler-specific directives

12. What is the difference between #include <> and #include ""?

Answer:

- **#include <>**: Searches for header files in system directories
- **#include ""**: Searches in current directory first, then system directories

13. What are header files?

Answer: Header files contain function declarations, macro definitions, and data type definitions. They have .h extension and are included using #include directive. Examples: stdio.h, stdlib.h, string.h

14. What are operators in C?

Answer: Operators are symbols that perform operations on operands:

- **Arithmetic**: +, -, *, /, %
- **Relational**: <, >, <=, >=, ==, !=

- **Logical**: &&, ||, !
- **Bitwise**: &, |, ^, ~, <<, >>
- **Assignment**: =, +=, -=, *=, /=
- **Unary**: ++, --, sizeof

15. What is the difference between ++i and i++?

Answer:

- **++i** (pre-increment): Increments i first, then returns the new value
- **i++** (post-increment): Returns current value of i, then increments

```
int i = 5;
int a = ++i; // a = 6, i = 6
int b = i++; // b = 6, i = 7
```

16. What are control statements in C?

Answer: Control statements control the flow of program execution:

- **Conditional**: if, if-else, switch
- **Looping**: for, while, do-while
- **Jump**: break, continue, goto, return

17. What is the difference between while and do-while loops?

Answer:

- **while**: Condition checked before loop execution (entry-controlled)
- **do-while**: Condition checked after loop execution (exit-controlled), executes at least once

18. What are functions in C?

Answer: Functions are reusable blocks of code that perform specific tasks. They help in modular programming and code reusability.

```
return_type function_name(parameters) {
    // function body
    return value;
}
```

19. What is the difference between call by value and call by reference?

Answer:

- **Call by value**: Copies the value of arguments to function parameters
- **Call by reference**: Passes the address of arguments to function parameters

```
void byValue(int x) { x = 10; }
void byReference(int *x) { *x = 10; }
```

20. What are arrays in C?

Answer: Arrays are collections of elements of the same data type stored in contiguous memory locations. Elements are accessed using indices starting from 0.

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

21. What is the difference between arrays and pointers?

Answer:

- **Arrays**: Fixed-size collection, name represents base address
- **Pointers**: Variables that store addresses, can be reassigned
- Array name is a constant pointer to the first element

22. What are strings in C?

Answer: Strings are arrays of characters terminated by null character ('\0'). C doesn't have a built-in string data type.

```
char str[] = "Hello";
char str2[6] = {'H', 'e', 'l', 'l', 'o', '\0'};
```

23. What are pointers in C?

Answer: Pointers are variables that store memory addresses of other variables. They provide indirect access to memory.

```
int x = 10;
int *ptr = &x;  // ptr stores address of x
```

24. What is the difference between malloc() and calloc()?

Answer:

- **malloc()**: Allocates memory block, doesn't initialize
- **calloc()**: Allocates memory and initializes to zero

```
int *p1 = malloc(sizeof(int) * 5);
int *p2 = calloc(5, sizeof(int));
```

25. What is dynamic memory allocation?

Answer: Dynamic memory allocation allows programs to allocate memory during runtime using functions like malloc(), calloc(), realloc(), and free().

26. What is the purpose of free() function?

Answer: free() deallocates memory previously allocated by malloc(), calloc(), or realloc() to prevent memory leaks.

27. What are structures in C?

Answer: Structures are user-defined data types that group related variables of different data types under a single name.

```
struct Student {
    char name[50];
    int age;
    float marks;
};
```

28. What is the difference between structure and union?

Answer:

- **Structure**: All members have separate memory locations
- **Union**: All members share the same memory location, only one member can be accessed at a time

29. What is typedef in C?

Answer: typedef creates aliases for existing data types, making code more readable and manageable.

```
typedef int Integer;
```

typedef struct Student Student_t;

30. What are enumeration (enum) in C?

Answer: Enumerations are user-defined data types consisting of named integer constants.

enum Days {MON, TUE, WED, THU, FRI, SAT, SUN};

31. What is the sizeof operator?

Answer: sizeof is a compile-time operator that returns the size of a data type or variable in bytes.

printf("%zu", sizeof(int)); // Prints size of int

32. What are format specifiers?

Answer: Format specifiers define the type of data to be printed or read:

- %d: Integer
- %f: Float
- %c: Character
- %s: String
- %p: Pointer address

33. What is the difference between printf() and scanf()?

Answer:

- **printf()**: Outputs formatted data to stdout
- **scanf()**: Reads formatted input from stdin

34. What are escape sequences?

Answer: Escape sequences are special characters preceded by backslash:

- \n: Newline
- \t: Tab
- \: Backslash
- ": Double quote
- \0: Null character

35. What is the difference between getchar() and getch()?

Answer:

- **getchar()**: Reads character from stdin, requires Enter key
- **getch()**: Reads character immediately without Enter (non-standard)

36. What are macros in C?

Answer: Macros are preprocessor directives that define constants or code snippets that are replaced during preprocessing.

```
#define PI 3.14159
#define MAX(a,b) ((a) > (b) ? (a) : (b))
```

37. What is the difference between macros and functions?

Answer:

- **Macros**: Text replacement, no function call overhead, preprocessed
- **Functions**: Actual function calls, type checking, compiled

38. What is recursion in C?

Answer: Recursion is when a function calls itself. It requires a base case to stop the recursive calls.

```
int factorial(int n) {
    if (n <= 1) return 1;
    return n * factorial(n - 1);
}
```

39. What are the advantages and disadvantages of recursion?

Answer: Advantages:

- Elegant and clean code
- Natural for problems with recursive structure

Disadvantages:

- Higher memory usage (stack)
- Slower execution
- Risk of stack overflow

40. What is the difference between local and global variables?

Answer:

- **Local**: Declared inside functions, limited scope, destroyed after function ends
- **Global**: Declared outside functions, accessible throughout program, exist until program ends

41. What are command line arguments?

Answer: Command line arguments are parameters passed to main() function when program is executed.

```
int main(int argc, char *argv[]) {
    // argc: argument count
    // argv: argument vector (array of strings)
}
```

42. What is the difference between exit() and return?

Answer:

- **return**: Returns control to calling function
- **exit()**: Terminates entire program immediately

43. What are file operations in C?

Answer: File operations include:

- Opening files (fopen)
- Reading (fread, fscanf)
- Writing (fwrite, fprintf)
- Closing (fclose)
- Seeking (fseek, ftell)

44. What is the difference between text and binary files?

Answer:

- **Text files**: Human-readable, platform-specific line endings
- **Binary files**: Machine-readable, exact byte-by-byte storage

45. What is the difference between fgets() and gets()?

Answer:

- **fgets()**: Safer, specifies buffer size, reads from specified stream
- **gets()**: Unsafe, no buffer size check, deprecated

46. What are the different file opening modes?

Answer:

- "r": Read only
- "w": Write only (truncates existing file)
- "a": Append
- "r+": Read and write
- "w+": Read and write (truncates)
- "a+": Read and append

47. What is the difference between fputc() and fputs()?

Answer:

- **fputc()**: Writes a single character to file
- **fputs()**: Writes a string to file

48. What is the NULL pointer?

Answer: NULL is a pointer that doesn't point to any valid memory location. It's typically defined as 0 or (void*)0.

49. What is a void pointer?

Answer: void pointer is a generic pointer that can point to any data type. It must be cast to specific type before dereferencing.

```
void *ptr;
int x = 10;
ptr = &x;
printf("%d", *(int*)ptr);
```

50. What is the difference between int *p and int (*p)?

Answer:

- **int *p*: p is a pointer to int
- **int (*p)**: p is a pointer (could be to function returning int)

Intermediate Level Questions

51. What are function pointers in C?

Answer: Function pointers are pointers that point to functions instead of variables. They enable dynamic function calls and callback mechanisms.

```
int add(int a, int b) { return a + b; }
int (*func_ptr)(int, int) = add;
int result = func_ptr(5, 3);
```

52. What is the difference between const int *p and int *const p?

Answer:

- **const int p*: Pointer to constant integer (can't modify value)
- **int const p*: Constant pointer to integer (can't modify pointer)
- **const int const p*: Constant pointer to constant integer

53. What are dangling pointers?

Answer: Dangling pointers point to memory locations that have been freed or are no longer valid. They can cause undefined behavior.

```
int *p = malloc(sizeof(int));
free(p);
// p is now a dangling pointer
```

54. What is a memory leak?

Answer: Memory leak occurs when dynamically allocated memory is not freed, causing the program to consume more memory over time.

55. What is the difference between stack and heap memory?

Answer:

- **Stack**: Automatic memory, LIFO, fast access, limited size
- **Heap**: Dynamic memory, slower access, larger size, manual management

56. What are wild pointers?

Answer: Wild pointers are uninitialized pointers that point to arbitrary memory locations.

```
int *p; // Wild pointer - uninitialized
```

57. What is pointer arithmetic?

Answer: Pointer arithmetic allows mathematical operations on pointers:

- Addition/Subtraction: Moves pointer by n elements
- Comparison: Compares memory addresses
- Difference: Number of elements between pointers

58. What is the difference between array and pointer declarations?

Answer:

```
int arr[10];    // Array declaration - allocates memory
int *ptr;       // Pointer declaration - doesn't allocate memory for data
```

59. What are multi-dimensional arrays?

Answer: Arrays with more than one dimension. Memory is allocated in row-major order.

```
int matrix[3][4];  // 2D array
int cube[2][3][4]; // 3D array
```

60. How do you pass a 2D array to a function?

Answer: Multiple ways:

```
void func1(int arr[][4], int rows);
void func2(int arr[3][4]);
void func3(int (*arr)[4], int rows);
```

61. What is the volatile keyword?

Answer: volatile tells the compiler that a variable's value may change unexpectedly, preventing optimization.

volatile int flag; // Value may change by hardware/interrupt

62. What is the register keyword?

Answer: register suggests storing a variable in CPU register for faster access. It's a hint to the compiler.

register int counter;

63. What are bit fields in structures?

Answer: Bit fields allow packing several variables into a single byte/word.

```
struct Flags {
   unsigned int flag1 : 1;
   unsigned int flag2 : 1;
   unsigned int value : 6;
};
```

64. What is structure padding and packing?

Answer:

- **Padding**: Compiler adds extra bytes for memory alignment
- **Packing**: Removing padding to save space using #pragma pack

65. What are unions and when to use them?

Answer: Unions allow different data types to share the same memory location. Used for:

- Memory conservation
- Type punning
- Variant data types

66. What is the difference between struct and typedef struct?

Answer:

```
struct Point { int x, y; };        // Need 'struct' keyword
typedef struct { int x, y; } Point; // Can use Point directly
```

67. What are nested structures?

Answer: Structures within structures.

```
struct Address {
   char city[20];
   int pincode;
};
struct Person {
   char name[30];
   struct Address addr;
};
```

68. What is the difference between #define and const?

Answer:

- **#define**: Preprocessor replacement, no type checking
- **const**: Compile-time constant, type-safe, scope-aware

69. What are variadic functions?

Answer: Functions that accept variable number of arguments using ellipsis (...).

```
#include <stdarg.h>
int sum(int count, ...) {
   va_list args;
   va_start(args, count);
   // Process arguments
   va_end(args);
}
```

70. What is the difference between parameter and argument?

Answer:

- **Parameter**: Variable in function definition
- **Argument**: Actual value passed to function

71. What are inline functions?

Answer: Functions marked with inline keyword are expanded at the point of call to reduce function call overhead.

72. What is the difference between library functions and user-defined functions?

Answer:

- **Library functions**: Pre-written functions (printf, scanf)
- **User-defined functions**: Functions written by programmer

73. What is function overloading? Is it possible in C?

Answer: Function overloading (same name, different parameters) is not supported in C. It's available in C++.

74. What are callback functions?

Answer: Functions passed as arguments to other functions, called back later.

```
void process(int arr[], int size, int (*callback)(int)) {
    for(int i = 0; i < size; i++)
        arr[i] = callback(arr[i]);
}
```

75. What is the difference between shallow copy and deep copy?

Answer:

- **Shallow copy**: Copies pointer values, not the pointed data
- **Deep copy**: Copies both pointer and the pointed data

76. What are the different types of errors in C?

Answer:

- **Syntax errors**: Incorrect syntax
- **Logical errors**: Wrong program logic
- **Runtime errors**: Errors during execution
- **Linker errors**: Errors during linking phase

77. What is the difference between compilation and interpretation?

Answer:

- **Compilation**: Translates entire source code to machine code before execution
- **Interpretation**: Translates and executes code line by line

78. What are the phases of compilation?

Answer:

1. Preprocessing
2. Compilation (to assembly)
3. Assembly (to object code)
4. Linking (to executable)

79. What is conditional compilation?

Answer: Compiling code based on preprocessor conditions.

```
#ifdef DEBUG
   printf("Debug mode\n");
#endif
```

80. What is the difference between #if and #ifdef?

Answer:

- **#ifdef**: Checks if macro is defined
- **#if**: Evaluates expression

81. What are static functions?

Answer: Functions with static storage class are visible only within the file where they're defined.

82. What is the difference between static and extern?

Answer:

- **static**: Internal linkage, visible only in current file
- **extern**: External linkage, visible across files

83. What are the different ways to initialize an array?

Answer:

```
int arr1[5] = {1, 2, 3, 4, 5};     // Complete initialization
int arr2[5] = {1, 2};              // Partial initialization
int arr3[] = {1, 2, 3};            // Size determined by initializer
int arr4[5] = {0};                 // All elements to 0
```

84. What is array decay?

Answer: When an array is passed to a function, it decays to a pointer to its first element.

85. What are the limitations of arrays?

Answer:

- Fixed size
- No bounds checking
- Can't return arrays from functions directly
- No built-in functions for operations

86. What is the difference between strcpy() and strncpy()?

Answer:

- **strcpy()**: Copies entire string, no length limit
- **strncpy()**: Copies up to n characters, safer

87. What are the different string functions in C?

Answer:

- strlen(): String length
- strcpy(): Copy string
- strcat(): Concatenate strings
- strcmp(): Compare strings
- strchr(): Find character
- strstr(): Find substring

88. What is the difference between sprintf() and printf()?

Answer:

- **printf()**: Prints to stdout
- **sprintf()**: Prints to string buffer

89. What are the different types of loops in C?

Answer:

- **for**: Known number of iterations
- **while**: Condition-controlled
- **do-while**: Executes at least once

90. What is an infinite loop and how to create one?

Answer: Loop that never terminates:

```
for(;;) { }          // Infinite for loop
while(1) { }         // Infinite while loop
do { } while(1);     // Infinite do-while loop
```

91. What is the difference between break and continue?

Answer:

- **break**: Exits the loop completely
- **continue**: Skips current iteration, continues with next

92. What is the goto statement?

Answer: goto transfers control to a labeled statement. Generally discouraged due to poor readability.

```
goto label;
label:
   printf("Jumped here\n");
```

93. What are the storage classes and their properties?

Answer:

- **auto**: Automatic storage, local scope, initialized with garbage
- **register**: Register storage, local scope, fast access
- **static**: Static storage, retains value, initialized with 0
- **extern**: External linkage, global scope

94. What is the difference between call by value and call by address?

Answer:

- **Call by value**: Passes copy of value
- **Call by address**: Passes memory address

95. What are the advantages of using functions?

Answer:

- Code reusability
- Modularity
- Easy testing and debugging
- Reduced code size
- Better organization

96. What is the main() function?

Answer: Entry point of C program. Execution starts from main().

```
int main() { }
int main(void) { }
int main(int argc, char *argv[]) { }
```

97. What are the different return types of main()?

Answer:

- **int**: Returns exit status
- **void**: No return value (not recommended)

98. What is the difference between local and global scope?

Answer:

- **Local scope**: Variables accessible only within the block
- **Global scope**: Variables accessible throughout the program

99. What is name mangling?

Answer: Process of encoding function names to include type information. Not done in C (unlike C++).

100. What are the different types of programming paradigms?

Answer:

- **Procedural**: Step-by-step execution (C follows this)
- **Object-oriented**: Based on objects and classes
- **Functional**: Based on functions
- **Declarative**: What to do, not how

Advanced Level Questions

101. What is the difference between `fflush(stdout)` and `fflush(stdin)`?

Answer:

- `fflush(stdout)`: Forces the output stream to be written immediately, ensuring data is displayed
- `fflush(stdin)`: Behavior is undefined in C standard, but some implementations clear the input buffer

printf("Hello");
fflush(stdout); // Ensures "Hello" is displayed immediately

102. What are the different ways to pass arrays to functions?

Answer:

1. **Array notation**: `void func(int arr[])`
2. **Pointer notation**: `void func(int *arr)`
3. **Sized array**: `void func(int arr[10])`
4. **Multi-dimensional**: `void func(int arr[][5])`

103. What is the difference between `memcpy()` and `memmove()`?

Answer:

- `memcpy()`: Copies memory blocks, undefined behavior if source and destination overlap
- `memmove()`: Safely handles overlapping memory regions

char str[] = "Hello World";
memmove(str + 2, str, 5); // Safe for overlapping regions

104. Write a program to implement a stack using arrays.

Answer:

#include <stdio.h>
#define MAX 100

```
struct Stack {
    int arr[MAX];
    int top;
};

void push(struct Stack *s, int value) {
    if (s->top == MAX - 1) {
        printf("Stack Overflow\n");
        return;
    }
    s->arr[++s->top] = value;
}

int pop(struct Stack *s) {
    if (s->top == -1) {
        printf("Stack Underflow\n");
        return -1;
    }
    return s->arr[s->top--];
}
```

105. What is the difference between `putchar()` and `puts()`?

Answer:

- `putchar()`: Outputs a single character
- `puts()`: Outputs a string and automatically adds a newline

```
putchar('A');       // Outputs: A
puts("Hello");      // Outputs: Hello\n
```

106. What are the different types of errors in C programming?

Answer:

1. **Compile-time errors**: Syntax errors, type mismatches
2. **Link-time errors**: Undefined references, missing libraries
3. **Run-time errors**: Division by zero, segmentation faults
4. **Logic errors**: Incorrect algorithm implementation

107. What is the difference between `size_t` and `int`?

Answer:

- `size_t`: Unsigned integer type, used for sizes and counts
- `int`: Signed integer type, can be negative
- `size_t` is guaranteed to hold the size of any object in bytes

108. Write a program to reverse a string without using library functions.

Answer:

```
#include <stdio.h>

void reverseString(char str[]) {
    int start = 0;
    int end = 0;

    // Find length
    while (str[end] != '\0') end++;
    end--;

    // Reverse
    while (start < end) {
        char temp = str[start];
        str[start] = str[end];
        str[end] = temp;
        start++;
        end--;
    }
}
```

109. What is the difference between `#include <stdio.h>` and `#include "stdio.h"`?

Answer:

- `#include <stdio.h>`: Searches system/standard directories first
- `#include "stdio.h"`: Searches current directory first, then system directories

110. What are the different ways to initialize a structure?

Answer:

```
struct Point {
```

```
    int x, y;
};

// Method 1: At declaration
struct Point p1 = {10, 20};

// Method 2: Using designated initializers
struct Point p2 = {.x = 10, .y = 20};

// Method 3: After declaration
struct Point p3;
p3.x = 10;
p3.y = 20;
```

111. What is the difference between `++*p` and `*++p`?

Answer:

- `++*p`: Increments the value pointed to by p
- `*++p`: Increments the pointer p, then dereferences it

```
int arr[] = {1, 2, 3};
int *p = arr;
++*p;    // arr[0] becomes 2
*++p;    // p points to arr[1], returns 2
```

112. Write a program to find the second largest element in an array.

Answer:

```
#include <stdio.h>

int findSecondLargest(int arr[], int n) {
    int first = arr[0], second = -1;

    for (int i = 1; i < n; i++) {
        if (arr[i] > first) {
            second = first;
            first = arr[i];
        } else if (arr[i] > second && arr[i] != first) {
            second = arr[i];
        }
```

```
    }
    return second;
}
```

113. What is the difference between `static` and `global` variables?

Answer:

- **Global variables**: Accessible from any file, external linkage
- **Static variables**: Accessible only within the file where declared, internal linkage

114. What are the different memory segments in a C program?

Answer:

1. **Text/Code Segment**: Executable code
2. **Data Segment**: Initialized global and static variables
3. **BSS Segment**: Uninitialized global and static variables
4. **Heap**: Dynamic memory allocation
5. **Stack**: Local variables and function calls

115. Write a program to implement binary search.

Answer:

```
#include <stdio.h>

int binarySearch(int arr[], int n, int target) {
    int left = 0, right = n - 1;

    while (left <= right) {
        int mid = left + (right - left) / 2;

        if (arr[mid] == target)
            return mid;
        else if (arr[mid] < target)
            left = mid + 1;
        else
            right = mid - 1;
    }
    return -1;
}
```

116. What is the difference between `realloc()` and `calloc()`?

Answer:

- `realloc()`: Resizes previously allocated memory block
- `calloc()`: Allocates memory and initializes to zero

```
int *ptr1 = calloc(5, sizeof(int));    // Allocates and initializes
int *ptr2 = realloc(ptr1, 10 * sizeof(int)); // Resizes memory
```

117. What are the different ways to declare a function?

Answer:

```
// Function declaration
int add(int a, int b);

// Function definition
int add(int a, int b) {
    return a + b;
}

// Function pointer declaration
int (*funcPtr)(int, int);
```

118. Write a program to check if a string is a palindrome.

Answer:

```
#include <stdio.h>
#include <string.h>

int isPalindrome(char str[]) {
    int start = 0;
    int end = strlen(str) - 1;

    while (start < end) {
        if (str[start] != str[end])
            return 0;
        start++;
        end--;
    }
```

```
    return 1;
}
```

119. What is the difference between `struct` and `union` in terms of memory?

Answer:

- **struct**: Each member has its own memory location, total size is sum of all members
- **union**: All members share the same memory location, size is that of the largest member

120. What are the different types of inheritance in C? (Trick Question)

Answer: C does not support inheritance as it's not an object-oriented language. Inheritance is a feature of C++ and other OOP languages.

121. Write a program to swap two numbers without using a third variable.

Answer:

```
#include <stdio.h>

void swap(int *a, int *b) {
    // Method 1: Using arithmetic
    *a = *a + *b;
    *b = *a - *b;
    *a = *a - *b;

    // Method 2: Using XOR
    // *a = *a ^ *b;
    // *b = *a ^ *b;
    // *a = *a ^ *b;
}
```

122. What is the difference between `signed` and `unsigned` data types?

Answer:

- **signed**: Can store both positive and negative values
- **unsigned**: Can store only positive values, larger positive range

```
signed int a = -100;     // Can be negative
```

```
unsigned int b = 100;    // Only positive
```

123. What are the different ways to read a file in C?

Answer:

1. **Character by character**: `fgetc()`
2. **Line by line**: `fgets()`
3. **Formatted input**: `fscanf()`
4. **Block reading**: `fread()`

124. Write a program to count the number of words in a string.

Answer:

```
#include <stdio.h>

int countWords(char str[]) {
    int count = 0;
    int inWord = 0;

    for (int i = 0; str[i] != '\0'; i++) {
        if (str[i] != ' ' && str[i] != '\t' && str[i] != '\n') {
            if (!inWord) {
                count++;
                inWord = 1;
            }
        } else {
            inWord = 0;
        }
    }
    return count;
}
```

125. What is the difference between `malloc()` and `new` operator?

Answer:

- `malloc()`: C function, allocates memory, returns void pointer
- `new`: C++ operator, allocates and initializes memory, returns typed pointer
- Note: `new` is not available in C

126. What are the different types of sorting algorithms? Implement bubble sort.

Answer: Types: Bubble sort, Selection sort, Insertion sort, Merge sort, Quick sort, Heap sort

```
void bubbleSort(int arr[], int n) {
    for (int i = 0; i < n - 1; i++) {
        for (int j = 0; j < n - i - 1; j++) {
            if (arr[j] > arr[j + 1]) {
                int temp = arr[j];
                arr[j] = arr[j + 1];
                arr[j + 1] = temp;
            }
        }
    }
}
```

127. What is the difference between `const char *p` and `char * const p`?

Answer:

- `const char *p`: Pointer to constant character (can't modify the character)
- `char * const p`: Constant pointer to character (can't modify the pointer)

128. Write a program to find the GCD of two numbers.

Answer:

```
#include <stdio.h>

int gcd(int a, int b) {
    if (b == 0)
        return a;
    return gcd(b, a % b);
}

// Iterative version
int gcdIterative(int a, int b) {
    while (b != 0) {
        int temp = b;
        b = a % b;
        a = temp;
```

```
    }
    return a;
}
```

129. What are the different ways to handle errors in C?

Answer:

1. **Return codes**: Functions return error codes
2. **Global error variables**: `errno`
3. **Assertions**: `assert()` macro
4. **Exception handling**: Not available in C (available in C++)

130. Write a program to implement a queue using arrays.

Answer:

```
#include <stdio.h>
#define MAX 100

struct Queue {
    int arr[MAX];
    int front, rear;
};

void enqueue(struct Queue *q, int value) {
    if (q->rear == MAX - 1) {
        printf("Queue Overflow\n");
        return;
    }
    if (q->front == -1) q->front = 0;
    q->arr[++q->rear] = value;
}

int dequeue(struct Queue *q) {
    if (q->front == -1 || q->front > q->rear) {
        printf("Queue Underflow\n");
        return -1;
    }
    return q->arr[q->front++];
}
```

131. What is the difference between `++i++` and `++(i++)`?

Answer: Both are invalid expressions in C. The `++` operator requires an lvalue, and `i++` returns an rvalue.

132. Write a program to convert decimal to binary.

Answer:

```
#include <stdio.h>

void decimalToBinary(int n) {
    if (n == 0) {
        printf("0");
        return;
    }

    int binary[32];
    int i = 0;

    while (n > 0) {
        binary[i] = n % 2;
        n = n / 2;
        i++;
    }

    for (int j = i - 1; j >= 0; j--) {
        printf("%d", binary[j]);
    }
}
```

133. What is the difference between `exit()` and `abort()`?

Answer:

- `exit()`: Normal program termination, calls cleanup functions
- `abort()`: Abnormal program termination, generates SIGABRT signal

134. Write a program to find the length of a string without using `strlen()`.

Answer:

```
#include <stdio.h>
```

```
int stringLength(char str[]) {
    int length = 0;
    while (str[length] != '\0') {
        length++;
    }
    return length;
}
```

135. What are the different ways to pass structures to functions?

Answer:

1. **Pass by value**: `void func(struct Point p)`
2. **Pass by reference**: `void func(struct Point *p)`
3. **Pass by address**: `void func(struct Point &p)` (C++ only)

136. Write a program to implement insertion sort.

Answer:

```
#include <stdio.h>

void insertionSort(int arr[], int n) {
    for (int i = 1; i < n; i++) {
        int key = arr[i];
        int j = i - 1;

        while (j >= 0 && arr[j] > key) {
            arr[j + 1] = arr[j];
            j--;
        }
        arr[j + 1] = key;
    }
}
```

137. What is the difference between `printf()` and `fprintf()`?

Answer:

- `printf()`: Writes to standard output (stdout)
- `fprintf()`: Writes to a specified file stream

```
printf("Hello World\n");              // To stdout
fprintf(stderr, "Error occurred\n");  // To stderr
```

138. Write a program to check if a number is prime.

Answer:

```
#include <stdio.h>
#include <math.h>

int isPrime(int n) {
    if (n <= 1) return 0;
    if (n <= 3) return 1;
    if (n % 2 == 0 || n % 3 == 0) return 0;

    for (int i = 5; i * i <= n; i += 6) {
        if (n % i == 0 || n % (i + 2) == 0)
            return 0;
    }
    return 1;
}
```

139. What is the difference between `strcat()` and `strncat()`?

Answer:

- `strcat()`: Concatenates entire second string to first
- `strncat()`: Concatenates at most n characters from second string

```
char dest[20] = "Hello ";
strcat(dest, "World");        // "Hello World"
strncat(dest, "123", 2);      // "Hello World12"
```

140. Write a program to implement selection sort.

Answer:

```
#include <stdio.h>

void selectionSort(int arr[], int n) {
    for (int i = 0; i < n - 1; i++) {
        int minIdx = i;
```

```
        for (int j = i + 1; j < n; j++) {
            if (arr[j] < arr[minIdx])
                minIdx = j;
        }
        if (minIdx != i) {
            int temp = arr[i];
            arr[i] = arr[minIdx];
            arr[minIdx] = temp;
        }
    }
}
```

141. What are the different ways to initialize a pointer?

Answer:

```
int x = 10;
int *ptr1 = &x;           // Initialize with address
int *ptr2 = NULL;         // Initialize with NULL
int *ptr3 = (int*)0;      // Initialize with 0
int *ptr4 = malloc(sizeof(int)); // Initialize with dynamic memory
```

142. Write a program to find the transpose of a matrix.

Answer:

```
#include <stdio.h>

void transpose(int matrix[3][3], int result[3][3]) {
    for (int i = 0; i < 3; i++) {
        for (int j = 0; j < 3; j++) {
            result[j][i] = matrix[i][j];
        }
    }
}
```

143. What is the difference between `calloc()` and `malloc()` in terms of initialization?

Answer:

- `malloc()`: Allocates memory but doesn't initialize (contains garbage values)
- `calloc()`: Allocates memory and initializes all bytes to zero

144. Write a program to implement linear search.

Answer:

```
#include <stdio.h>

int linearSearch(int arr[], int n, int target) {
    for (int i = 0; i < n; i++) {
        if (arr[i] == target)
            return i;
    }
    return -1;
}
```

145. What is the difference between `break` and `return` statements?

Answer:

- `break`: Exits the nearest enclosing loop or switch statement
- `return`: Exits the current function and optionally returns a value

146. Write a program to check if two strings are anagrams.

Answer:

```
#include <stdio.h>
#include <string.h>

int areAnagrams(char str1[], char str2[]) {
    int len1 = strlen(str1);
    int len2 = strlen(str2);

    if (len1 != len2) return 0;

    int count[256] = {0};

    for (int i = 0; i < len1; i++) {
        count[str1[i]]++;
        count[str2[i]]--;
    }
```

```
    for (int i = 0; i < 256; i++) {
        if (count[i] != 0) return 0;
    }
    return 1;
}
```

147. What is the difference between `fseek()` and `ftell()`?

Answer:

- `fseek()`: Sets the file position indicator to a specific location
- `ftell()`: Returns the current file position indicator

```
fseek(file, 0, SEEK_END);  // Move to end of file
long size = ftell(file);   // Get current position (file size)
```

148. Write a program to find the sum of digits of a number.

Answer:

```
#include <stdio.h>

int sumOfDigits(int n) {
    int sum = 0;
    while (n > 0) {
        sum += n % 10;
        n /= 10;
    }
    return sum;
}
```

149. What are the different ways to declare and initialize arrays?

Answer:

```
// Declaration
int arr1[5];

// Declaration with initialization
int arr2[5] = {1, 2, 3, 4, 5};
```

```
// Partial initialization
int arr3[5] = {1, 2};

// Size determined by initializer
int arr4[] = {1, 2, 3, 4, 5};

// All elements to same value
int arr5[5] = {0};
```

150. Write a program to implement matrix multiplication.

Answer:

```
#include <stdio.h>

void multiplyMatrices(int first[2][3], int second[3][2], int result[2][2]) {
    for (int i = 0; i < 2; i++) {
        for (int j = 0; j < 2; j++) {
            result[i][j] = 0;
            for (int k = 0; k < 3; k++) {
                result[i][j] += first[i][k] * second[k][j];
            }
        }
    }
}
```

Expert Level Bonus Questions

Bonus: What are the key differences between C89, C99, and C11 standards?

Answer:

- **C89/C90**: Original ANSI C standard
- **C99**: Added inline functions, variable-length arrays, complex numbers
- **C11**: Added threading support, improved Unicode support, atomic operations

Bonus: Explain the concept of undefined behavior in C with examples.

Answer: Undefined behavior occurs when the C standard doesn't specify what should happen:

- **Buffer overflow**: `int arr[5]; arr[10] = 5;`
- **Null pointer dereference**: `int *p = NULL; *p = 5;`
- **Signed integer overflow**: `INT_MAX + 1`
- **Use after free**: `free(ptr); *ptr = 5;`
