C Programming

Complete Learning Notes

Concepts • Examples • Programs • Interview Preparation

SkillovaHub

Student-Friendly Programming Notes

Contents

C Introduction

What is C Programming?

C is a general-purpose programming language created in the 1970s. It is widely used for system programming, applications and embedded systems. C is fast, portable and powerful, and allows programmers to work closely with memory and hardware.

Features of C Language

- Easy to understand and straightforward
- Fast and efficient
- Portable across platforms
- Supports structured programming
- Widely used in system programming

Basic C Program Example

```
#include <stdio.h>

int main()
{
    printf("Hello, World!");
    return 0;
}
```

Output: Hello, World!

Code Explanation

- #include <stdio.h> includes the standard input/output library.
- int main() is the starting point of program execution.
- printf() displays output.
- return 0 terminates the program successfully.

Conclusion

C is an important programming language and a strong starting point for learning programming logic.

C Variables

What are Variables in C?

A variable is a named storage location used to hold data that can be used and modified during program execution. Each variable has a name, a data type and a value.

Rules for Naming Variables

- Use letters, digits and underscore.
- Do not start with a digit.
- Do not use spaces.
- Do not use C keywords.
- Use meaningful names.

Valid: age, student_name, totalMarks, num1

Invalid: 1age, student name, int, total-marks

Variable Declaration

Syntax:
data_type variable_name;

Examples:
int age;
float salary;
char grade;

Variable Initialization

Syntax:
data_type variable_name = value;

Examples:
int age = 20;
float salary = 25000.50;
char grade = 'A';

Example Program

```
#include <stdio.h>

int main()
{
    int age = 20;
    printf("Age = %d", age);
    return 0;
}
```

Output: Age = 20

Key Points

- Variables store data.
- Every variable has a data type.
- Variables must follow naming rules.
- Variables can be initialized at declaration.

C Data Types

What are Data Types in C?

Data types define what kind of value a variable can store. Common basic types are int, float, char and double.

Why Data Types are Important?

- Define the type of stored value.
- Help the compiler manage memory.
- Improve accuracy and performance.
- Make programs easier to understand.

Basic Data Types

- int → integer values

- float → decimal values
- char → a single character
- double → decimal values with higher precision

Memory Size

- char → commonly 1 byte
- int → commonly 4 bytes
- float → commonly 4 bytes
- double → commonly 8 bytes

Example Program

```
#include <stdio.h>

int main()
{
    int age = 20;
    float salary = 25000.50;
    char grade = 'A';

    printf("Age = %d\n", age);
    printf("Salary = %.2f\n", salary);
    printf("Grade = %c", grade);
    return 0;
}
```

Output:
Age = 20
Salary = 25000.50
Grade = A

Key Points

- Data types define the kind of data stored.
- Different types generally use different amounts of memory.
- Choosing a suitable type improves program efficiency.

C Operators

What are Operators in C?

Operators are symbols used to perform operations on variables and values, including calculations, comparisons, logical decisions, assignment and bit manipulation.

Types of Operators

- Arithmetic
- Relational
- Logical
- Assignment
- Increment and Decrement
- Bitwise
- Conditional (Ternary)

Arithmetic Operators

- + Addition
- - Subtraction
- * Multiplication
- / Division
- % Modulus

Relational Operators

- == Equal to
- != Not equal to
- > Greater than
- < Less than
- >= Greater than or equal to
- <= Less than or equal to

Logical Operators

- && Logical AND
- || Logical OR
- ! Logical NOT

Assignment Operators

- = Assign
- += Add and assign
- -= Subtract and assign
- *= Multiply and assign
- /= Divide and assign

Increment and Decrement

- ++ increases a value by 1.
- -- decreases a value by 1.

Bitwise Operators

- & AND
- | OR
- ^ XOR
- ~ NOT
- << Left shift
- >> Right shift

Conditional Operator

Syntax:
condition ? expression1 : expression2;
It is a compact form of if-else.

Example Program

```
#include <stdio.h>

int main()
{
    int a = 10, b = 5;
    printf("Addition = %d\n", a + b);
    printf("Subtraction = %d\n", a - b);
    printf("Multiplication = %d\n", a * b);
    printf("Division = %d\n", a / b);
    printf("Modulus = %d\n", a % b);
    return 0;
}
```

Output:
Addition = 15
Subtraction = 5
Multiplication = 50
Division = 2
Modulus = 0

C Decision Making

What is Decision Making in C?

Decision-making statements execute different blocks of code depending on whether conditions are true or false.

Types

- if
- if...else
- Nested if
- else-if ladder
- switch
- Conditional (?:) operator

if Statement

```
Executes a block only when the condition is true.

if(condition)
{
    // statements
}
```

if...else Statement

Provides two execution paths: one for a true condition and one for a false condition.

Nested if

Places one if statement inside another for dependent or multiple conditions.

else-if Ladder

Checks multiple conditions from top to bottom and executes the first true condition.

switch

Selects one block from multiple options using case labels. break is commonly used to stop after a matching case.

Example Program

```
#include <stdio.h>

int main()
{
    int marks = 78;
    if(marks >= 90)
        printf("Grade A");
    else if(marks >= 75)
        printf("Grade B");
    else if(marks >= 50)
        printf("Grade C");
    else
        printf("Fail");
    return 0;
}
```

Output: Grade B

C Loops

What are Loops in C?

Loops repeatedly execute a block of code while a condition permits. They reduce repetition and improve efficiency.

Types of Loops

- for loop
- while loop
- do-while loop

for Loop

```
Used when the iteration pattern is known.

for(initialization; condition; increment/decrement)
{
    // code
}
```

while Loop

```
Checks the condition before each iteration.

while(condition)
{
    // code
}
```

do-while Loop

```
Executes the body first and checks the condition afterward, so it executes at least once.

do
{
    // code
}
while(condition);
```

Example Program

```
#include <stdio.h>

int main()
{
    int i;
```

```
    for(i = 1; i <= 5; i++)
        printf("%d\n", i);
    return 0;
}
```

Output:

1
2
3
4
5

Key Points

- Loops reduce repetition.
- for is commonly used for counted iterations.
- while checks before execution.
- do-while executes at least once.

C Functions

What are Functions in C?

A function is a block of code that performs a specific task. Functions improve reusability, readability and maintenance.

Types of Functions

- Library Functions
- User-Defined Functions

Library Functions

Predefined functions supplied by the C library, such as printf(), scanf(), strlen(), sqrt() and pow().

User-Defined Functions

```
Functions created by the programmer for specific tasks.

void greet()
{
    printf("Welcome to C Programming");
}
```

Example Program

```
#include <stdio.h>

void greet()
{
    printf("Welcome to C Programming");
}

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

Output: Welcome to C Programming

Advantages

- Reduce code duplication.

- Improve readability.
- Make debugging easier.
- Increase code reusability.
- Organize large programs into modules.

C Arrays

What are Arrays in C?

An array is a collection of elements of the same data type stored in consecutive memory locations. Indexing starts from 0.

One-Dimensional Array

Example:
int marks[5] = {80, 75, 90, 85, 95};
The first element is marks[0] and the last is marks[4].

Two-Dimensional Array

Stores values in rows and columns.

int marks[2][3] = {{80,75,90},{85,95,88}};

Multi-Dimensional Array

An array with more than two dimensions, useful for advanced applications.

int marks[2][2][2];

Example Program

```
#include <stdio.h>

int main()
{
    int marks[5] = {80, 75, 90, 85, 95};
    int i;
    for(i = 0; i < 5; i++)
        printf("%d\n", marks[i]);
    return 0;
}
```

Output:
80
75
90
85
95

C Strings

What are Strings in C?

A string is a collection of characters stored in a character array and terminated by the null character \0.

Character Array

char name[] = "SkillovaHub";

String Pointer

char *name = "SkillovaHub";

A string literal should not be modified through an unsuitable pointer.

Example Program

```
#include <stdio.h>

int main()
{
    char name[] = "SkillovaHub";
    printf("String = %s", name);
    return 0;
}
```

Output: String = SkillovaHub

Key Points

- Strings are stored as character arrays.
- Every C string ends with \0.
- %s displays a string.
- String functions support common text operations.

C Pointers

What are Pointers in C?

A pointer is a variable that stores the memory address of another variable. Pointers are important for arrays, functions, dynamic memory and data structures.

Basic Example

int age = 25;
int *ptr = &age;

&age gives the address; ptr stores it; *ptr accesses the value at that address.

Null Pointer

int *ptr = NULL;
A null pointer does not point to a valid object.

Void Pointer

void *ptr = &age;
A void pointer can hold an address of different object types and must be handled with the appropriate type.

Wild Pointer

int *ptr;
An uninitialized pointer may contain an indeterminate address and should not be dereferenced.

Example Program

```
#include <stdio.h>

int main()
```

```
{
    int age = 25;
    int *ptr = &age;
    printf("Value of age = %d\n", age);
    printf("Address of age = %p\n", (void*)&age);
    printf("Value stored in pointer = %d\n", *ptr);
    return 0;
}
```

Output includes the value 25 and its memory address.

C Structures

What are Structures in C?

A structure is a user-defined data type that groups different types of data under one name.

Example

```
struct Student
{
    int id;
    char name[20];
    float marks;
};
```

Nested Structure

A structure containing another structure as a member.

Array of Structures

An array whose elements are structure variables, useful for storing multiple records.

Self-Referential Structure

A structure containing a pointer to another structure of the same type, commonly used in linked lists.

Example Program

```
#include <stdio.h>

struct Student
{
    int id;
    char name[20];
    float marks;
};

int main()
{
    struct Student s = {101, "Rahul", 89.5};
    printf("ID: %d\n", s.id);
    printf("Name: %s\n", s.name);
    printf("Marks: %.1f\n", s.marks);
    return 0;
}
```

Output:
ID: 101
Name: Rahul
Marks: 89.5

C Unions

What are Unions in C?

A union is a user-defined type in which different members share the same memory location. This can reduce memory use when only one member is needed at a time.

Example

```
union Data
{
    int id;
    float marks;
    char grade;
};
```

Important Point

Writing a value to one union member uses the shared storage, so the previous member's representation may be overwritten.

Anonymous Union

An anonymous union has no union tag name and its members can be accessed directly in supported C environments.

Example Program

```
#include <stdio.h>

union Data
{
    int id;
    float marks;
    char grade;
};

int main()
{
    union Data d;
    d.id = 101;
    printf("ID: %d\n", d.id);
    d.marks = 89.5;
    printf("Marks: %.1f\n", d.marks);
    d.grade = 'A';
    printf("Grade: %c\n", d.grade);
    return 0;
}
```

Output:
ID: 101
Marks: 89.5
Grade: A

C File Handling

What is File Handling in C?

File handling is the process of creating, opening, reading, writing, updating and closing files so data can be stored persistently.

File Pointer

```
FILE *fp;
fp = fopen("student.txt", "w");
```

Reading

fopen("student.txt", "r") opens an existing file for reading.

Writing

fopen("student.txt", "w") creates a file if needed and overwrites existing contents.

Appending

fopen("student.txt", "a") adds new data at the end while retaining existing data.

Example Program

```
#include <stdio.h>

int main()
{
    FILE *fp = fopen("student.txt", "w");
    if(fp == NULL)
    {
        printf("File could not be opened.");
        return 1;
    }
    fprintf(fp, "Welcome to SkillovaHub");
    fclose(fp);
    printf("Data written successfully.");
    return 0;
}
```

Output: Data written successfully.

Key Functions

- fopen()
- fclose()
- fprintf()
- fscanf()
- fgetc()
- fputc()
- fgets()
- fputs()

C Dynamic Memory Allocation

What is Dynamic Memory Allocation?

Dynamic Memory Allocation allocates and releases memory during program execution. The standard functions are malloc(), calloc(), realloc() and free().

malloc()

```
Allocates a specified number of bytes. The allocated storage is not initialized.

int *ptr = (int *)malloc(sizeof(int));
```

calloc()

Allocates memory for multiple elements and initializes the allocated bytes to zero.

int *ptr = (int *)calloc(5, sizeof(int));

realloc()

Changes the size of an existing allocation.

ptr = (int *)realloc(ptr, 10 * sizeof(int));

free()

Releases dynamically allocated memory.

free(ptr);

Example Program

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

int main()
{
    int *ptr = (int *)malloc(sizeof(int));
    if(ptr == NULL) return 1;
    *ptr = 100;
    printf("Value = %d\n", *ptr);
    free(ptr);
    return 0;
}
```

Output: Value = 100

C Storage Classes

What are Storage Classes?

Storage classes describe scope, lifetime and storage behavior of variables. Commonly taught storage classes are auto, register, static and extern.

auto

Default storage class for local variables.

register

Requests register storage for a local variable; the compiler may ignore the request.

static

Preserves a variable's value between function calls.

extern

Declares a variable that is defined elsewhere, commonly used across source files.

Example Program

```
#include <stdio.h>

void display()
{
    static int count = 0;
    count++;
    printf("Count = %d\n", count);
}

int main()
{
```

```
    display(); display(); display();
    return 0;
}
```

Output:
Count = 1
Count = 2
Count = 3

C Header Files

What are Header Files?

Header files contain declarations, macros, constants and definitions that can be shared across source files. #include is used to include them.

Common Header Files

- stdio.h → input/output
- stdlib.h → utilities and dynamic memory
- string.h → string handling
- math.h → mathematical functions
- ctype.h → character handling

User-Defined Header

```
#include "myheader.h"

Programmers can create their own reusable header files.
```

Example Program

```
#include <stdio.h>

int main()
{
    printf("Welcome to SkillovaHub\n");
    printf("Header Files Example");
    return 0;
}
```

Output:
Welcome to SkillovaHub
Header Files Example

C Preprocessor Directives

What are Preprocessor Directives?

Preprocessor directives are processed before compilation. They begin with # and are used for header inclusion, symbolic constants/macros and conditional compilation.

#include

#include <stdio.h> includes declarations needed for standard input/output functions.

#define

#define MAX 100 creates a symbolic constant or macro.

Conditional Compilation

#ifdef, #ifndef, #if, #else and #endif can control which sections are compiled.

Example Program

```
#include <stdio.h>
#define PI 3.14

int main()
{
    float radius = 5;
    float area = PI * radius * radius;
    printf("Area = %.2f", area);
    return 0;
}
```

Output: Area = 78.50

C Typedef

What is Typedef?

typedef creates an alias for an existing data type. It does not create a new underlying data type.

Basic Type

```
typedef int Integer;
Integer age = 25;
```

With Structures

```
typedef struct
{
    int id;
    char name[20];
} Student;

Student s1;
```

With Pointers

```
typedef int *IntPointer;
IntPointer ptr;
```

Example Program

```
#include <stdio.h>
typedef int Integer;

int main()
{
    Integer age = 25;
    printf("Age = %d", age);
    return 0;
}
```

Output: Age = 25

C Enumeration (enum)

What is Enumeration?

An enumeration is a user-defined type consisting of named integer constants. By default, the first enumerator is 0 and subsequent enumerators increase by 1.

Simple Enumeration

enum Color { Red, Green, Blue };

Assigned Values

enum Status { Success = 1, Warning = 5, Error = 10 };

Enumeration Variable

```
enum Day { Sunday, Monday, Tuesday };
enum Day today = Monday;
```

Example Program

```
#include <stdio.h>

enum Day { Sunday, Monday, Tuesday, Wednesday };

int main()
{
    enum Day today = Tuesday;
    printf("Today's value = %d", today);
    return 0;
}
```

Output: Today's value = 2

C Recursion

What is Recursion?

Recursion is a technique in which a function calls itself. A base case is required to stop further calls.

Direct Recursion

A function directly calls itself.

Indirect Recursion

One function calls another function which eventually calls the first.

Tail Recursion

The recursive call is the last operation in the function.

Example Program

```
#include <stdio.h>

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

int main()
{
    printf("Factorial = %d", factorial(5));
    return 0;
}
```

Output: Factorial = 120

C Bitwise Operators

What are Bitwise Operators?

Bitwise operators work on the binary representation of integer values.

AND (&)

5 (0101) & 3 (0011) = 1 (0001).

OR (|)

5 (0101) | 3 (0011) = 7 (0111).

XOR (^)

5 (0101) ^ 3 (0011) = 6 (0110).

Shift Operators

5 << 1 gives 10 in this example. 5 >> 1 gives 2 for this positive integer example.

Example Program

```
#include <stdio.h>

int main()
{
    int a = 5, b = 3;
    printf("AND = %d\n", a & b);
    printf("OR = %d\n", a | b);
    printf("XOR = %d\n", a ^ b);
    printf("Left Shift = %d\n", a << 1);
    printf("Right Shift = %d\n", a >> 1);
    return 0;
}
```

Output:
AND = 1
OR = 7
XOR = 6
Left Shift = 10
Right Shift = 2

C Command Line Arguments

What are Command Line Arguments?

Command line arguments allow values to be passed to a program when it is launched from the command line. They are received through argc and argv.

Example

```
#include <stdio.h>

int main(int argc, char *argv[])
{
    printf("Number of Arguments = %d", argc);
    return 0;
}
```

argc stores the argument count; argv stores the argument strings. argv[0] normally contains the program name.

Why Important?

- Allow input without modifying source code.

- Support automation and scripting.
- Make programs flexible and reusable.
- Help with file and data processing.

C Interview Questions

Introduction

C interview preparation should cover fundamentals, programming logic, memory management, pointers, functions, arrays, structures, strings and file handling.

30 Common Questions

- What is C Programming? — A general-purpose procedural language widely used for systems and embedded programming.
- What are basic data types? — int, char, float and double.
- What is a variable? — A named memory location used to store data.
- What is a constant? — A value intended not to change during execution.
- What is a function? — A block of code that performs a specific task.
- What is recursion? — A function calling itself until a base case is reached.
- What is a pointer? — A variable storing a memory address.
- Structure vs Union? — Structure members have separate storage; union members share storage.
- What is an array? — Same-type elements stored in contiguous memory locations.
- What is a string? — A character sequence terminated by \0.
- What is a header file? — A file containing declarations and related definitions.
- What is main()? — The entry point of a C program.
- What are storage classes? — auto, register, static and extern.
- What is dynamic memory allocation? — Runtime memory allocation using malloc(), calloc(), realloc() and free().
- What is a null pointer? — A pointer that does not point to a valid object.
- What is a file pointer? — A FILE pointer used for file operations.
- malloc vs calloc? — malloc does not initialize allocated storage; calloc initializes it to zero.
- What does break do? — Terminates a loop or switch.
- What does continue do? — Skips to the next loop iteration.
- while vs do-while? — while checks before execution; do-while checks after the body.
- What is sizeof()? — An operator that returns the size in bytes of a type or object.
- What is a dangling pointer? — A pointer referring to memory that is no longer valid.
- What is a macro? — A preprocessor substitution defined using #define.
- What is an infinite loop? — A loop that does not terminate.
- ++i vs i++? — Pre-increment changes the value before use; post-increment uses the old value first.
- What is a segmentation fault? — A runtime error caused by invalid memory access.
- Why is C called a middle-level language? — It combines low-level memory access with high-level structured programming features.
- What is typedef? — An alias for an existing type.

• What is enum? — A user-defined type of named integer constants.

• What is file handling? — Persistent file creation, reading, writing, updating and closing operations.

Top 10 Frequently Asked C Interview Programs

• Factorial

• Fibonacci Series

• Prime Number

• Palindrome Number

• Armstrong Number

• Reverse a Number

• Swap Two Numbers

• Matrix Addition

• String Reverse

• Sorting an Array

Interview Tips

• Understand fundamentals before advanced concepts.

• Practice programs without copying.

• Learn pointers and memory management carefully.

• Revise loops, arrays, strings and functions.

• Practice explaining program logic.

• Practice debugging.

• Prepare both theory and programming questions.