Computer Training Institute ®
TECH FUSION

Your Shortcut to Success – Free C programming Notes Inside!

Next page

What is C Programming?

C is a general-purpose, procedural programming language developed by Dennis Ritchie in 1972 at Bell Labs. It's powerful, fast, and forms the base for many modern languages like C++, Java, and Python.

Why Learn C?

- It helps you understand how memory works.
- Many systems and embedded programs are written in C.
- It builds strong programming fundamentals.

Features of C:

- Simple and efficient
- Fast execution
- Low-level memory access
- Procedural language
- Portable (runs on many systems)

Setting Up Environment

🔷 On Windows:

- Use Code::Blocks, Turbo C++, or VS Code + GCC
- Download GCC Compiler via MinGW
- Use online compilers like replit.com or onlinegdb.com

🔷 On Mac/Linux:

- Install GCC using brew install gcc or sudo apt install build-essential

Basic Structure of a C Program

```c
#include <stdio.h>

int main() {
    // This is a simple C program
    printf("Hello, World!");
    return 0;
}
```

- #include <stdio.h>: Preprocessor command to include standard I/O library
- int main(): Entry point of the program
- printf(): Outputs text to the screen
- return 0;: Indicates successful execution

Compilation & Execution Process

- Write Code → .c file
- Compile → Converts to machine code using a compiler like gcc
- Execute → Runs the compiled program
- Example (Command line)

```
gcc hello.c -o hello
./hello
```

Practical Tasks

- Practice 1: Install a C compiler (or use an online one)
- Practice 2: Write and run your first program

```
#include <stdio.h>

int main() {
    printf("Learning C is fun!
\n");
    return 0;
}
```

- Practice 3: Modify the program
- Try printing your name.
  Print multiple lines using \n.

What are Data Types?

Data types define the type of data a variable can store. C is a statically typed language, which means the type of a variable must be declared before using it.

🔷 Primary Data Types in C

Data Type	Description	Size (in bytes)	Format Specifier
int	Integer numbers	2 or 4	%d
float	Decimal numbers	4	%f
double	Double-precision float	8	%lf
char	Single character	1	%c

Variables in C

A variable is a named memory location used to store a value.

Syntax:

```
data_type variable_name;
```

Example:

```
int age;
float marks;
char grade;
```

Variable Declaration & Initialization

```
int age = 20;
float pi = 3.14;
char grade = 'A';
```

Constants in C

Using const keyword:

```
const float PI = 3.14;
```

Using #define:

```
#define PI 3.14
```

Type Conversion

C automatically converts data types in expressions when needed. This is called implicit conversion.

Example:

```
int a = 5;
float b = 2.5;
float c = a + b;
// a is implicitly converted to float
```

You can also do explicit conversion (typecasting):

```
int a = 10;
float b = (float)a;
```

Practice 1: Declare variables of different types

```
#include <stdio.h>

int main() {
    int age = 25;
    float weight = 65.5;
    char initial = 'O';
    printf("Age: %d\nWeight: %.1f\nInitial:
%c\n", age, weight, initial);
    return 0;
}
```

Practice 2: Try constants

```
#include <stdio.h>
#define PI 3.1416

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

Practice 3: Play with typecasting

```
#include <stdio.h>

int main() {
    int x = 10, y = 3;
    float result = (float)x / y;
    printf("Result: %.2f\n", result);
    return 0;
}
```

What are Operators in C?

Operators are symbols that perform operations on variables and values.

🔷 Types of Operators

Data Type	Example Operators	Purpose
Arithmetic	+, -, *, /, %	Perform basic mathematical operations
Relational	==, ≠, >, <, ⩾, ⩽	Compare two values (true/false result)
Logical	&&, `	Perform logical operations with control structures
Assignment	=, +=, -=, *=, /=, %=	Assign values to variables
Bitwise	&, `	work directly on the binary (bit-by-bit) level of numbers
Increment/Decrement	++, --	Increase or decrease value by 1
Conditional (Ternary)	?, :	Short form of if-else

Next page

Detailed Examples

Arithmetic Operators

```
int a = 10, b = 5;
int sum = a + b;     // sum = 15
int diff = a - b;    // diff = 5
int prod = a * b;    // prod = 50
int quot = a / b;    // quot = 2
int rem = a % b;     // rem = 0
```

Relational Operators

```
int a = 5, b = 10;
printf("%d", a > b);  // 0 (false)
printf("%d", a < b);  // 1 (true)
```

Logical Operators

```
int a = 5, b = 10;
printf("%d", (a < b) && (b > 0)); // 1 (true)
printf("%d", (a > b) || (b > 0)); // 1 (true)
printf("%d", !(a < b));           // 0 (false)
```

Assignment Operators

```
int a = 10;
a += 5; // a = a + 5 → a = 15
a *= 2; // a = a * 2 → a = 30
```

Next page

Detailed Examples

Bitwise Operators

```
int a = 5, b = 3;
printf("%d\n", a & b);   // AND → 1
printf("%d\n", a | b);   // OR → 7
printf("%d\n", a ^ b);   // XOR → 6
```

Increment/Decrement Operators

```
int a = 5;
a++;  // a = 6
a--;  // a = 5
```

Conditional (Ternary) Operator

```
int a = 5, b = 10;
int max = (a > b) ? a : b;
printf("%d", max); // 10
```

Practice 1: Arithmetic Operations

```
#include <stdio.h>

int main() {
    int x = 12, y = 4;
    printf("Sum = %d\n", x + y);
    printf("Difference = %d\n", x - y);
    printf("Product = %d\n", x * y);
    printf("Quotient = %d\n", x / y);
    printf("Remainder = %d\n", x % y);
    return 0;
}
```

Practice 2: Compare two numbers

```
#include <stdio.h>

int main() {
    int a, b;
    printf("Enter two numbers: ");
    scanf("%d%d", &a, &b);
    printf("Is a greater than b? %d\n", a > b);
    printf("Are they equal? %d\n", a == b);
    return 0;
}
```

Practice 3: Use Logical and Ternary Operator

```
#include <stdio.h>

int main() {
    int age;
    printf("Enter your age: ");
    scanf("%d", &age);
    (age ⩾ 18) ? printf("Eligible to vote\n")
: printf("Not eligible to vote\n");
    return 0;
}
```

Next page

What are Control Statements?

Control statements direct the flow of a program based on certain conditions or repetitions. Without them, a program would just run line-by-line without making decisions or repeating tasks.

🔷 if statement

```
if (condition) {
    // code if condition is true
}
```

Example:

```
int age = 18;
if (age ⩾ 18) {
    printf("Eligible to vote\n");
}
```

🔷 if-else statement

```
if (condition) {
    // code if condition is true
} else {
    // code if condition is false
}
```

Next page

🔷 else-if ladder

```
if (condition1) {
    // code block 1
} else if (condition2) {
    // code block 2
} else {
    // code block 3
}
```

🔷 switch statement

```
switch (expression) {
    case value1:
        // code block
        break;
    case value2:
        // code block
        break;
    default:
        // default code block
}
```

Jumping Statements

🔷 break
- Immediately exits a loop or a switch case.

🔷 continue
- Skips the current iteration and moves to the next.

🔷 goto
- Directs the program flow to a labeled part of the code (⚠ not recommended for good practice).

Practice 1: Simple if-else

```c
#include <stdio.h>

int main() {
    int marks;
    printf("Enter your marks: ");
    scanf("%d", &marks);

    if (marks ⩾ 50) {
        printf("You passed!\n");
    } else {
        printf("You failed.\n");
    }
    return 0;
}
```

Practice 2: else-if ladder (Grade System)

```c
#include <stdio.h>

int main() {
    int marks;
    printf("Enter your marks: ");
    scanf("%d", &marks);

    if (marks ⩾ 90) {
        printf("Grade A\n");
    } else if (marks ⩾ 75) {
        printf("Grade B\n");
    } else if (marks ⩾ 50) {
        printf("Grade C\n");
    } else {
        printf("Fail\n");
    }
    return 0;
}
```

Practice 3: switch-case example

```c
#include <stdio.h>

int main() {
    int day;
    printf("Enter day number (1-7): ");
    scanf("%d", &day);

    switch (day) {
        case 1: printf("Monday\n"); break;
        case 2: printf("Tuesday\n"); break;
        case 3: printf("Wednesday\n"); break;
        case 4: printf("Thursday\n"); break;
        case 5: printf("Friday\n"); break;
        case 6: printf("Saturday\n"); break;
        case 7: printf("Sunday\n"); break;
        default: printf("Invalid day!\n");
    }
    return 0;
}
```

Next page

What are Loops?

Loops allow you to repeat a block of code multiple times until a condition is false. They are essential for tasks like printing a pattern, processing arrays, or running a program until user exits.

Loop Type	Description
for loop	Used when the number of iterations is known
while loop	Used when the number of iterations is unknown and depends on a condition
do-while loop	Similar to while loop but executes at least once

1. for Loop

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

Example:

```
for (int i = 1; i ⩽ 5; i++) {
    printf("%d ", i);
}
```

Next page

2. while Loop

```
while (condition) {
    // code to be repeated
}
```

Example:

```
int i = 1;
while (i ⩽ 5) {
    printf("%d ", i);
    i++;
}
```

3. do-while Loop

```
do {
    // code to be repeated
} while (condition);
```

Example:

```
int i = 1;
do {
    printf("%d ", i);
    i++;
} while (i ⩽ 5);
```

Next page

When to Use What:

Use Case	Loop to Use
Known number of iterations	for loop
Unknown condition beforehand	while loop
Need to run at least once	do-while loop

Practice 1: Print 1 to 10 using all 3 loops

```c
#include <stdio.h>
int main() {
    int i;
    // For loop
    printf("Using for loop:\n");
    for (i = 1; i ⩽ 10; i++) {
        printf("%d ", i);
    }
    // While loop
    printf("\nUsing while loop:\n");
    i = 1;
    while (i ⩽ 10) {
        printf("%d ", i);
        i++;
    }
    // Do-while loop
    printf("\nUsing do-while loop:\n");
    i = 1;
    do {
        printf("%d ", i);
        i++;
    } while (i ⩽ 10);
    return 0;
}
```

Practice 2: Sum of first N numbers

```c
#include <stdio.h>

int main() {
    int n, sum = 0;
    printf("Enter a number: ");
    scanf("%d", &n);

    for (int i = 1; i ⩽ n; i++) {
        sum += i;
    }

    printf("Sum = %d\n", sum);
    return 0;
}
```

Practice 3: Table of a number using while loop

```c
#include <stdio.h>

int main() {
    int num, i = 1;
    printf("Enter a number: ");
    scanf("%d", &num);

    while (i ⩽ 10) {
        printf("%d x %d = %d\n", num, i, num *
i);
        i++;
    }

    return 0;
}
```

What is an Array in C?

An array is a collection of variables of the same data type, stored at contiguous memory locations, and accessed using an index.

Think of it as a row of boxes where each box holds one item (a number, character, etc.) of the same type.

Types of arrays in c	Description
1D Array	Linear collection of elements
2D Array	Like a table with rows and columns
Multidimensional Array	Array with more than 2 dimensions

1D Array (One-Dimensional Array)

```
int numbers[5]; // declares an array of 5 integers

int numbers[5] = {1, 2, 3, 4, 5}; //Initialisation

printf("%d", numbers[2]); // Outputs 3 (index
starts from 0)
//Accessing elements
```

2D Array (Two-Dimensional Array)

```
int matrix[2][3]; // 2 rows, 3 columns
(Declaration)
```

Next page

2D Array (Two-Dimensional Array) Continued...

```
int matrix[2][3] = {
    {1, 2, 3},
    {4, 5, 6}      //Initialisation
};

printf("%d", matrix[1][2]); // Outputs 6
//Accessing elements
```

Practice 1: Input & Output of a 1D Array

```
#include <stdio.h>

int main() {
    int a[5], i;
    printf("Enter 5 numbers:\n");

    for (i = 0; i < 5; i++) {
        scanf("%d", &a[i]);
    }

    printf("You entered:\n");
    for (i = 0; i < 5; i++) {
        printf("%d ", a[i]);
    }

    return 0;
}
```

Next page

Practice 2: Find the Largest Element in Array

```c
#include <stdio.h>
int main() {
    int a[5], i, max;
    printf("Enter 5 numbers:\n");
    for (i = 0; i < 5; i++) {
        scanf("%d", &a[i]);
    }
    max = a[0];
    for (i = 1; i < 5; i++) {
        if (a[i] > max) {
            max = a[i];
        }
    }
    printf("Largest number is %d\n", max);
    return 0;
}
```

Practice 3: Sum of all elements in a 2D Array

```c
#include <stdio.h>

int main() {
    int a[2][2], i, j, sum = 0;

    printf("Enter 4 elements (2x2 matrix):\n");
    for (i = 0; i < 2; i++) {
        for (j = 0; j < 2; j++) {
            scanf("%d", &a[i][j]);
            sum += a[i][j];
        }
    }

    printf("Sum of elements = %d\n", sum);
    return 0;
}
```

💡 Tips:

- Arrays are zero-indexed in C.
- Always ensure you don’t exceed the array bounds.
- Use loops to efficiently work with arrays.

Next Page

What is a String in C?

A string is a sequence of characters terminated by a null character ('\0').
It is essentially a character array.

🔷 Declaring Strings

```
char name[10];          // can hold up to 9 characters + '\0'
char name[] = "Omkar";  // automatically adds '\0'
char name[6] = {'O','m','k','a','r','\0'};
// manual declaration
```

🔷 Common String Functions (from <string.h>)

Function	Description
strlen(s)	Returns length of string
strcpy(a, b)	Copies string b to string a
strcat(a, b)	Concatenates b to the end of a
strcmp(a, b)	Compares two strings

⚠ Important Notes

- Strings must be large enough to hold the null character.
- Functions from <string.h> must be used with care — they don't auto-check array size limits.

Practice 1: Input and Print a String

```c
#include <stdio.h>

int main() {
    char name[50];
    printf("Enter your name: ");
    scanf("%s", name);  // Note: scanf stops at
whitespace
    printf("Hello, %s!\n", name);
    return 0;
}
```

Practice 2: String Length and Copy

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

int main() {
    char str1[50], str2[50];
    printf("Enter a string: ");
    scanf("%s", str1);

    printf("Length = %lu\n", strlen(str1));

    strcpy(str2, str1);
    printf("Copied String: %s\n", str2);

    return 0;
}
```

Next page

Practice 1: Compare Strings

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

int main() {
    char s1[20], s2[20];
    printf("Enter first string: ");
    scanf("%s", s1);
    printf("Enter second string: ");
    scanf("%s", s2);
    if (strcmp(s1, s2) == 0)
        printf("Strings are equal.\n");
    else
        printf("Strings are not equal.\n");
    return 0;
}
```

🧠 Bonus Concepts:

- Use fgets() instead of gets() for safe input.
- String manipulation is heavily used in text-based applications and file processing.

Next page

What is a Function?

A function is a block of reusable code that performs a specific task.
Functions help you organize code, avoid repetition, and make programs modular.

🔷 Types of Functions

Type	Example
Library Functions	printf(), scanf(), strlen()
User-Defined Functions (custom made)	void greet(), int add(int, int)

Function Syntax

```
return_type function_name(parameters) {
    // code
    return value; // if not void
}
```

Function Declaration (Prototype)

Place this above main() if the function is defined after main():

```
int add(int, int);
```

Next page

What is a Function?

```c
#include <stdio.h>

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

int main() {
    int sum = add(5, 10);    // function call
    printf("Sum = %d\n", sum);
    return 0;
}
```

🔷Function Categories Based on Arguments and Return

Category	Example Use Case
No arguments, no return value	void greet()
Arguments, no return value	void display(int x)
No arguments, with return value	int getNumber()
Arguments and return value	int multiply(int a, int b)

Next page

Practice 1: No argument, no return

```c
#include <stdio.h>

void greet() {
    printf("Hello! Welcome to C Programming.\n");
}

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

Practice 2: Argument, no return

```c
#include <stdio.h>

void square(int x) {
    printf("Square: %d\n", x * x);
}

int main() {
    square(4);
    return 0;
}
```

Next page

Practice 3: No argument, with return

```c
#include <stdio.h>

int getNumber() {
    int n;
    printf("Enter a number: ");
    scanf("%d", &n);
    return n;
}

int main() {
    int num = getNumber();
    printf("You entered: %d\n", num);
    return 0;
}
```

Practice 4: Argument and return

```c
#include <stdio.h>

int multiply(int a, int b) {
    return a * b;
}

int main() {
    int result = multiply(5, 3);
    printf("Result = %d\n", result);
    return 0;
}
```

What is a Pointer?

A pointer is a variable that stores the memory address of another variable.
Instead of storing a value directly, it points to a location in memory where the value is stored.

🔷 Why Use Pointers?

> To access and manipulate memory directly
> For dynamic memory allocation
> For arrays and strings
> For function arguments (pass by reference)
> Crucial in data structures like linked lists, trees, etc.

Declaring and Using a Pointer

```
int a = 10;
int *p = &a;  // 'p' stores the address of 'a'
```

Symbol	Meaning
&a	Address of variable a
*p	Value at the address stored in p (dereferencing)

Next page

Example: Basic Pointer Use

```
#include <stdio.h>

int main() {
    int a = 5;
    int *ptr = &a;

    printf("Value of a = %d\n", a);
    printf("Address of a = %p\n", &a);
    printf("Pointer ptr = %p\n", ptr);
    printf("Value at ptr = %d\n", *ptr);

    return 0;
}
```

🔷 Pointer to Pointer

```
int a = 5;
int *p = &a;
int **q = &p;

printf("%d\n", **q); // prints 5
```

Next page

Practice 1: Access value and address

```c
#include <stdio.h>

int main() {
    int num = 100;
    int *ptr = &num;

    printf("Value: %d\n", *ptr);
    printf("Address: %p\n", ptr);
    return 0;
}
```

Practice 2: Swap two numbers using pointers

```c
#include <stdio.h>

void swap(int *x, int *y) {
    int temp = *x;
    *x = *y;
    *y = temp;
}

int main() {
    int a = 10, b = 20;
    swap(&a, &b);
    printf("a = %d, b = %d\n", a, b);
    return 0;
}
```

Next page

What is a Structure?

A structure in C is a user-defined data type that allows grouping of variables of different data types under a single name.
It's used to represent complex data types like student records, employee info, etc.

🔷 Why Use Structures?

> To model real-world entities
> To organize related data
> To pass multiple values to functions
> Essential in data handling, file I/O, and data structures

Structure Syntax

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

Declaring and Using Structure Variables

```
struct Student {
    int id;
    char name[50];
    float marks;
} s1, s2;
```

Next page

🔹 Accessing Structure Members

```
s1.id = 101;
printf("%d", s1.id);
```

🔹 Using struct with scanf and printf

```
scanf("%d", &s1.id);
scanf("%s", s1.name);
scanf("%f", &s1.marks);
```

Practice 1: Create and Print Student Info

```
#include <stdio.h>

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

int main() {
    struct Student s;

    printf("Enter student ID, name, and
marks:\n");
    scanf("%d %s %f", &s.id, s.name,
&s.marks);

    printf("\nStudent Details:\n");
    printf("ID: %d\n", s.id);
    printf("Name: %s\n", s.name);
    printf("Marks: %.2f\n", s.marks);

    return 0;
}
```

Practice 2: Array of Structures

```
#include <stdio.h>

struct Employee {
    int id;
    char name[50];
};

int main() {
    struct Employee e[3];

    for (int i = 0; i < 3; i++) {
        printf("Enter ID and name of
employee %d: ", i + 1);
        scanf("%d %s", &e[i].id,
e[i].name);
    }

    printf("\nEmployee List:\n");
    for (int i = 0; i < 3; i++) {
        printf("ID: %d, Name: %s\n",
e[i].id, e[i].name);
    }

    return 0;
}
```

Next page

Practice 3: Passing Structure to Function

```
#include <stdio.h>

struct Point {
    int x, y;
};

void display(struct Point p) {
    printf("x = %d, y = %d\n", p.x,
p.y);
}

int main() {
    struct Point p1 = {3, 4};
    display(p1);
    return 0;
}
```

Next page

What is File Handling?

File handling in C allows you to create, read, write, and manipulate files stored on the disk — enabling data storage between program executions.

Why File Handling?

> To store data permanently
> To handle large volumes of input/output
> For building systems like databases, logs, user data, etc.

🔷 Basic File Operations

Operation	Function Used
Create/Open	fopen()
Read	fread(), fgets(), fscanf()
Write	fprintf(), fputs(), fwrite()
Close	fclose()

File Opening Modes

Mode	Description
"r"	Read (file must exist)
"w"	Write (creates new file or truncates)
"a"	Append (writes at the end)
"r+"	Read & Write
"w+"	Read & Write (creates or truncates)
"a+"	Read & Append

Opening a File

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

Closing a File

```
fclose(fp);
```

Next page

Practice 1: Write to a File

```c
#include <stdio.h>
int main() {
    FILE *fptr;
    fptr = fopen("output.txt", "w");
    if (fptr == NULL) {
        printf("Error opening file!
\n");
        return 1;
    }
    fprintf(fptr, "Hello, File Handling
in C!\n");
    fclose(fptr);
    printf("Data written successfully.
\n");
    return 0;
}
```

Practice 2: Read from a File

```c
#include <stdio.h>

int main() {
    FILE *fptr;
    char ch;
    fptr = fopen("output.txt", "r");
    if (fptr == NULL) {
        printf("File not found!\n");
        return 1;
    }
    while ((ch = fgetc(fptr)) ≠ EOF) {
        putchar(ch);
    }
    fclose(fptr);
    return 0;
}
```

Next page

Practice 3: Append Data to File

```c
#include <stdio.h>

int main() {
    FILE *fptr = fopen("output.txt",
"a");

    if (fptr == NULL) {
        printf("File not found!\n");
        return 1;
    }

    fprintf(fptr, "Appending this line.
\n");
    fclose(fptr);
    printf("Line appended successfully.
\n");
    return 0;
}
```

The End