



IAS YOUTH COMPUTER CENTRE

BASIC IN C PROGRAMMING

A Practical Beginner's Guide to C

IAS YOUTH COMPUTER CENTRE
88/1, Queen Apartment, 2nd Floor,
Dumdum Road, Near Life Line Medical Store,
Kolkata – 700030, West Bengal

Learn • Code • Practice • Build

About This Book

This beginner-friendly book introduces C programming from the fundamentals through practical programming exercises. It covers syntax, variables, input/output, operators, conditions, loops, functions, arrays, strings, pointers, structures, dynamic memory and file handling.

Learning method: Read the concept → type the example → compile → run → test with different inputs → improve the program.

Learning Outcomes

- Understand the basic structure of a C program.
- Declare and use variables and data types.
- Take input and produce formatted output.
- Use operators, conditions and loops.
- Create reusable functions.
- Work with arrays and strings.
- Understand pointers and structures.
- Use dynamic memory and basic file handling.
- Debug and test simple C programs.

Contents

Chapter	Topic
1	Introduction to C
2	Program Structure & Compilation
3	Variables & Data Types
4	Input & Output
5	Operators & Expressions
6	Decision Making
7	Loops
8	Functions
9	Arrays
10	Strings
11	Pointers
12	Structures & Unions
13	Dynamic Memory
14	File Handling
15	Preprocessor & Headers
16	Practical Programs
17	Debugging & Good Practices
18	Viva Questions
19	Glossary
20	Final Practical Test

Chapter 1: Introduction to C

C is a general-purpose programming language used to teach programming fundamentals and to build efficient software.

Why Learn C

- It develops structured problem-solving skills.
- It provides a foundation for understanding memory and computer systems.
- It is widely used in systems, embedded and performance-oriented programming.

Programming Basics

A program is a sequence of instructions. An algorithm describes the steps needed to solve a problem. A compiler translates C source code into an executable form.

Practical Activity

Write an algorithm for adding two numbers before writing the program.

Quick Revision

- What is C?
- What is an algorithm?
- What does a compiler do?
- Why is C useful for learning programming?

Chapter 2: Program Structure & Compilation

A simple C program contains a function such as `main()` and may include standard library headers.

Example

```
#include <stdio.h>

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

Key Parts

- `#include` can include declarations from a header.
- `main()` is the typical entry point of a C program.
- `printf()` displays formatted output.
- `return 0` indicates normal completion in a conventional main function.

Practical Activity

Type and run the Hello World program, then change the message.

Quick Revision

- What is `main()`?
- What is `#include`?
- What does `printf()` do?
- What is the compile-run cycle?

Chapter 3: Variables & Data Types

Variables store values. A data type specifies the kind of value a variable can represent.

Common Types

- int – integer values
- char – characters
- float – floating-point values
- double – higher-precision floating-point values
- void – absence of a value in applicable contexts

Declaration

Example: **int age;**. A value can then be assigned using **age = 20;**.

Constants

Constants represent values intended not to change through normal program logic; const variables and symbolic definitions are common approaches.

Practical Activity

Create variables for name/initial, age and percentage. Assign values and display them.

Quick Revision

- What is a variable?
- What is a data type?
- Name four C data types.
- What is a constant?

Chapter 4: Input & Output

Most programs accept data, process it and display a result.

Output

printf() displays text and formatted values.

Input

scanf() reads formatted input into variables. For many variable types, the address is supplied using &.

Format Specifiers

- %d – int
- %c – character
- %s – string
- %f – commonly used for floating-point formatting; exact specifier usage depends on the type and function.

Practical Activity

Write a program that reads two integers and displays their sum, difference and product.

Quick Revision

- What is input?
- What is output?
- What does scanf() do?
- Why are format specifiers needed?

Chapter 5: Operators & Expressions

Operators perform calculations, comparisons and logical operations.

Arithmetic

- + addition
- - subtraction
- * multiplication
- / division
- % remainder

Relational

- == equal
- != not equal
- > greater than
- < less than
- >= greater/equal
- <= less/equal

Logical

- && AND
- || OR
- ! NOT

Assignment

= assigns a value. Compound forms include +=, -=, *= and /=.

Expressions

Expressions combine values, variables and operators; precedence affects evaluation order.

Practical Activity

Build a mini calculator using arithmetic operators.

Quick Revision

- Name five arithmetic operators.
- Difference between = and ==?
- What does && mean?
- What is an expression?

Chapter 6: Decision Making

Conditional statements allow a program to choose actions based on conditions.

if

Runs a block when a condition is true.

if-else

Provides an alternative block when the condition is false.

else-if Ladder

Checks multiple conditions in sequence.

switch

Selects among cases based on an integral or compatible expression; break is commonly used to prevent unwanted fall-through.

Practical Activity

Read a number and report whether it is positive, negative or zero.

Quick Revision

- What is a condition?
- When is if used?
- What is else used for?
- When is switch useful?

Chapter 7: Loops

Loops repeat statements according to a condition or iteration rule.

for

Useful when initialization, condition and update are conveniently expressed together.

while

Checks the condition before each iteration.

do-while

Runs the body at least once before checking the condition.

Loop Control

- break can terminate a loop.
- continue can skip the remaining statements in the current iteration.

Practical Activity

Print numbers 1–20 using for and then rewrite using while.

Quick Revision

- Why use loops?
- Difference between while and do-while?
- What does break do?
- What does continue do?

Chapter 8: Functions

Functions divide programs into reusable, manageable units.

Function Parts

- Return type
- Name
- Parameters
- Body
- Return statement when needed

Prototype & Definition

A prototype declares a function before use; its definition contains the implementation.

Parameters & Return

Parameters pass information into a function; a return value sends a result back.

Practical Activity

Create `add(int a, int b)` that returns the sum.

Quick Revision

- Why use functions?
- What is a parameter?
- What is a return value?
- What is a prototype?

Chapter 9: Arrays

An array stores multiple elements of the same declared type.

Declaration

Example: `int marks[5];`.

Indexing

C arrays use zero-based indexing; the first element is index 0.

Traversal

Loops are commonly used to read or process array elements.

Multidimensional Arrays

Arrays can have multiple dimensions, such as rows and columns.

Practical Activity

Store five marks and calculate total and average.

Quick Revision

- What is an array?
- What is the first index?
- Why use loops with arrays?
- What is a 2D array?

Chapter 10: Strings

A C string is typically a character array terminated by a null character.

Character Arrays

Example: `char name[20];`

String Functions

string.h provides functions such as `strlen()`, `strcpy()`, `strcat()` and `strcmp()`. Use them with appropriate buffer sizes.

Input Safety

Choose input methods carefully and ensure character buffers are large enough.

Practical Activity

Read a short name and display its length using `strlen()`.

Quick Revision

- What is a C string?
- What terminates it?
- Name two string functions.
- Why is buffer size important?

Chapter 11: Pointers

A pointer stores an address and is central to arrays, functions and dynamic memory.

Address Operator

& obtains the address of an object.

Dereference

* accesses the value at an address when the pointer is valid.

Declaration

Example: `int *p;`

Uses

Pointers are used for indirect access, dynamic memory and passing addresses to functions.

Practical Activity

Create an integer and pointer, then print the value, address and dereferenced value.

Quick Revision

- What is a pointer?
- What does & do?
- What does * do for dereferencing?
- Why are pointers important?

Chapter 12: Structures & Unions

Structures group related data, including members of different types.

Structure

A student record can contain roll number, name and marks in one structure.

Member Access

The dot operator accesses members through an object; -> is used through a pointer to a structure.

Union

A union stores different members in the same memory area, with one member's stored value being meaningfully active at a time.

Practical Activity

Create a Student structure with roll, name and marks and display one record.

Quick Revision

- What is a structure?
- How are members accessed?
- What is a union?
- Why use structures?

Chapter 13: Dynamic Memory

Dynamic memory is requested during execution when size or lifetime needs are determined at runtime.

Functions

- `malloc()` allocates a block.
- `calloc()` allocates and initializes a block.
- `realloc()` changes the size of an allocation.
- `free()` releases allocated memory.

Header

These functions are declared in `stdlib.h`.

Good Practice

Check allocation results when appropriate and free memory when it is no longer needed.

Practical Activity

Allocate five integers with `malloc()`, fill them, print them and free the memory.

Quick Revision

- Why use dynamic memory?
- What does `malloc` do?
- What does `free` do?
- Which header is used?

Chapter 14: File Handling

File handling lets programs store and retrieve information beyond one execution.

FILE Pointer

A FILE pointer is used with standard C file functions.

Opening

fopen() opens a file using a mode such as read, write or append.

Reading/Writing

Functions include fprintf(), fscanf(), fgets(), fputs(), fgetc() and fputc().

Closing

fclose() closes an opened file.

Practical Activity

Write a message to a text file, close it, reopen it and display the contents.

Quick Revision

- What is a FILE pointer?
- What does fopen do?
- Why close a file?
- Name two file functions.

Chapter 15: Preprocessor & Headers

The preprocessor handles directives before the main compilation stage.

Common Directives

- `#include` – include header content
- `#define` – define a macro or symbolic replacement
- `#if/#ifdef` and related directives – conditional compilation

Headers

`stdio.h` provides standard input/output declarations; `string.h` provides common string functions; `stdlib.h` provides many utility and memory-allocation functions.

Macros

Macros are processed before ordinary compilation and should be used carefully.

Practical Activity

Use `#include` and define a simple symbolic constant with `#define`.

Quick Revision

- What is the preprocessor?
- What does `#include` do?
- What does `#define` do?
- Name two header files.

Chapter 16: Practical Programs

Combine the fundamentals through small programs.

Practice Set

- Calculator
- Even or odd checker
- Factorial
- Prime-number check
- Student result calculator
- Array sum/average/minimum/maximum

Method

Write the algorithm first, code it, compile, test normal inputs, test edge cases and improve the program.

Practical Activity

Choose three problems and solve them without copying a solution.

Quick Revision

- Why write algorithms first?
- What is an edge case?
- How should a program be tested?

Chapter 17: Debugging & Good Practices

Debugging is the process of finding and correcting errors.

Error Types

- Syntax error – violates language rules
- Runtime error – occurs during execution
- Logical error – program runs but produces an incorrect result

Good Practices

- Use meaningful names
- Indent consistently
- Keep functions focused
- Comment useful intent
- Test boundary inputs
- Compile frequently

Debugging Process

Read the error message → locate the problem → inspect values and assumptions → make a small change → compile → retest.

Practical Activity

Take a small program with an intentional error and classify it as syntax, runtime or logical before fixing it.

Quick Revision

- Name three error types.
- What is a logical error?
- Why use meaningful names?
- Why read compiler messages?

Chapter 18: Viva & Interview Questions

What is C?

A general-purpose programming language used for programming fundamentals and efficient software.

What is a variable?

A named storage location associated with a data type.

What is the difference between = and ==?

= assigns a value; == compares values for equality.

What is a loop?

A control structure that repeats code.

What is a function?

A named, reusable block of program logic.

What is an array?

A sequence of elements of the same declared type.

What is a string?

A null-terminated character sequence.

What is a pointer?

A variable that stores an address.

What is a structure?

A user-defined type grouping related members.

What is dynamic memory?

Memory requested during program execution.

What is file handling?

Using standard C functions to read and write persistent data.

Practice Questions

- Explain the structure of a C program.
- Compare for, while and do-while.
- Explain functions and return values.
- Explain arrays and strings.
- Explain pointers with a simple example.
- Compare structure and union.
- Explain malloc() and free().
- Describe a basic debugging process.

Chapter 19: C Programming Glossary

Term	Meaning
Algorithm	Step-by-step procedure for solving a problem.
Array	Sequence of elements of the same declared type.
Compiler	Tool that translates source code.
Data Type	Specifies the kind of value represented.
Function	Named reusable block of program logic.
Header File	File containing declarations/interfaces used by programs.
Loop	Control structure for repetition.
Operator	Symbol that performs an operation.
Pointer	Variable that stores an address.
String	Null-terminated sequence of characters.
Structure	User-defined type grouping related members.
Variable	Named storage location.
malloc	Function used to allocate dynamic memory.
free	Function used to release dynamic memory.
printf	Standard formatted output function.
scanf	Standard formatted input function.
fopen	Function used to open a file.

Chapter 20: Final Practical Test

Complete the following project independently. Write, compile, test and improve your programs.

Task A – Student Result Program

- Read student details and marks.
- Store marks in an array.
- Calculate total and average.
- Use at least one function.
- Use conditions to display a result category.
- Display a clear final report.

Task B – File Program

- Write a few records to a text file.
- Close the file properly.
- Reopen the file for reading.
- Display the stored records.

Task C – Programming Challenge

- Create a menu-driven calculator or number utility.
- Use functions and a loop.
- Validate important inputs.
- Add comments and meaningful names.
- Test normal and edge-case inputs.

Self Assessment

Skill	Can do	Need practice
Program structure	■	■
Variables & data types	■	■
Input & output	■	■
Operators	■	■
Conditions	■	■
Loops	■	■
Functions	■	■
Arrays	■	■
Strings	■	■
Pointers	■	■
Structures	■	■
Dynamic memory	■	■

Skill	Can do	Need practice
File handling	■	■
Debugging	■	■

Final Goal: Write and understand basic C programs, use control structures and functions, work with arrays, strings, pointers and structures, handle basic files and debug simple programs.



BASIC IN C PROGRAMMING

Learn • Code • Practice • Build

IAS YOUTH COMPUTER CENTRE

88/1, Queen Apartment, 2nd Floor,
Dumdum Road, Near Life Line Medical Store,
Kolkata – 700030, West Bengal