Parul® University

Programming for Problem Solving **(PPS)**

Chapter-1 Introduction to ‘C’ Programming:

Parul®
University

What is a Program?

- Program is a collection of instructions that performs a specific task when executed by a computer.
- **What are Programming Languages?**
- A programming language is a set of commands, instructions and other syntax use to create a software program. Programming languages are used in computer programming to implement algorithms.

Parul®
University

DIGITAL LEARNING CONTENT

Introduction to C Programming Language

- C is a **general-purpose, high-level** programming language, which means it can be used to write a wide range of programs—from operating systems to games.
- **High-level**: Easier for humans to read and write than machine code.
- **General-purpose**: Not limited to specific applications.
- **Low-level features**: Allows direct memory access using pointers.
- **Procedural**: Follows a step-by-step approach using functions.
- **Compiled**: Code is converted into machine language before execution.
- **Structured**: Code is divided into blocks (functions), making it easier to manage.

Common Usage: Operating systems, embedded systems, and compilers.

1. High Level Language

- **Meaning**:

  A high-level language provides abstraction from the hardware. This means you don’t need to worry about managing memory locations, CPU registers, or writing binary code.

Why it matters:

- You can write commands like printf(“Hello”) instead of complicated machine-level instructions.
- It uses English-like keywords (e.g., if, while, for, return).
- Easier to learn and debug than assembly or machine code.

2. General-Purpose

- **Meaning**:

  C is not restricted to solving a specific type of problem or limited to one domain.

  You can use it for a wide variety of applications.

Where it can be used:

- System software (OS, drivers)
- Embedded systems (firmware)
- Application software (compilers, databases)
- Scientific computing
- Games and graphics engines

Why it matters:

- You learn one language and can apply it almost anywhere.

3. Low-Level Features

- **Meaning**:

  Even though C is a high-level language, it gives access to low-level operations like memory manipulation through **pointers**.

What this lets you do:

- Access and modify memory directly using addresses.
- Write efficient and fast code.
- Communicate with hardware-level resources.

Why it matters:

- Critical for writing OS kernels, drivers, and performance-critical code.

4. Procedural

- **Meaning**:

  C follows the **procedure-oriented** approach. The program is divided into **functions or procedures** that run step-by-step.

Characteristics:

- Executes instructions in the order they are written.
- Uses main() as the starting point.
- Functions are used to divide tasks (e.g., input(), process(), output()).

Why it matters:

- Easier to write, understand, and maintain code.
- Encourages logical thinking and problem-solving through small steps.

5. Compiled

- **Meaning**:

  C is a **compiled language**, meaning the program you write (source code) is first converted into **machine code** by a compiler before it can be executed.

Steps:

1. Write code in .c file.
2. Compile it using a compiler like GCC.
3. The compiler converts it into .exe (Windows) or a.out (Linux).
4. Run the executable.

Why it matters:

- Faster execution than interpreted languages.
- Errors are caught during compilation.
- No need for the compiler to run the program again.

6. Structured

- **Meaning**:
  C supports **structured programming**, which means organizing code into **blocks**, mainly using **functions**, **loops**, **conditionals**, etc.

Benefits:

- Increases readability and reusability.
- Reduces errors and complexity.
- Makes debugging easier.

Why it matters:

- Students learn to break problems into smaller tasks (modular design).
- Helps in managing large codebases.

History of C Language

- Developed in **1972** by **Dennis Ritchie** at **Bell Labs**.
- Evolved from:
  - -**BCPL** → Basic Combined Programming Language
  - -**B** (developed by Ken Thompson)
  - **-C** (a successor to B)
- **UNIX OS** was originally written in assembly, then rewritten in **C** (1973), making it portable.
- Standardization:
  - -**ANSI C (1989)** – American National Standards Institute version
  - -**ISO C** – International Standardization, later versions like C99, C11, C18.

Application Areas of C

- C is used in various fields due to its performance and control over hardware:
- **System Programming**: Operating systems like UNIX, Linux kernels.
- **Embedded Systems**: Microcontrollers, firmware for smart devices.
- **Game Development**: Core engines and performance-critical parts.
- **Compilers/Interpreters**: GCC and others are written in C.
- **GUI Applications**: Though rare today, some interfaces still use C.
- **Database Systems**: MySQL is developed in C.
- **Network Programming**: Socket programming, protocol implementations.
- **Scientific & Engineering**: Simulations and numerical analysis.

1. System Programming

- Used for: Operating Systems, Kernels, Drivers, File Systems
- C was originally developed to write **UNIX**, and even today, parts of **Linux, Windows**, and **macOS** kernels are written in C.
- It gives access to **low-level operations** like memory and I/O ports.
- System programs need **fast execution**, **direct hardware access**, and **reliability**, all of which C provides.

2. Embedded Systems

- Used for: Microcontrollers, IoT devices, Smart appliances
- In embedded systems, memory and processing power are limited. C is ideal because it creates **compact and fast executables**.
- C can communicate directly with **hardware registers** and **sensors** via memory addresses.
- Used in **automobiles, washing machines, smart TVs, thermostats, medical devices**, etc.
- **Example**:
  Firmware of a microwave oven or heartbeat monitor uses C to control its operations.

3. Game Development

- Used for: Game engines, Physics calculations, Graphics rendering
- C (and C++) powers the **core engine** of many games because it's extremely fast.
- Games need **real-time performance**, especially in physics engines and rendering pipelines.
- C is often used to build **custom graphics engines**, **AI algorithms**, or **low-level optimizations**.
- **Example**:
  The Unreal Engine (originally) and early versions of games like **Doom** and **Quake** used C.

4. Compilers and Interpreters

- Used for: Building programming tools, language compilers
- Many popular compilers like **GCC (GNU Compiler Collection)** are written in C.
- C helps in building the compiler's **lexer, parser, optimizer, and code generator**.
- Interpreters for other languages also often rely on a C-written runtime.
- **Example**:
  Python’s interpreter(CPython) is written in C.

5. GUI Applications

- Used for: Creating desktop applications with graphical interfaces
- C was widely used for GUI-based applications in early Windows and Linux.
- Even though high-level languages (like Java, C#, Python) are used today, C libraries like **GTK** and **WinAPI** are still written in C.
- Developers may use C for the **backend logic** and then connect to a GUI frontend.
- **Example**:
  The **GIMP image editor** uses C and the **GTK+ toolkit**.

6. Database Systems

- Used for: Core development of relational databases
- C provides **fine control over memory and file systems**, which is essential for database engines.
- Since databases need high-speed processing for large data sets, C is an ideal choice.
- It also allows efficient **indexing**, **query parsing**, and **memory pooling**.
- **Example**:
  MySQL, one of the most widely used relational databases, is written in C and C++.

7. Network Programming

- Used for: Protocol implementation, server/client architecture
- Network programs often deal with **sockets**, **ports**, and **protocols (like TCP/IP)**.
- C provides **low-level socket access** and the ability to work closely with the OS's networking stack.
- Many **web servers, proxies, and firewalls** are built using C.
- **Example**:
  Nginx, a powerful web server used by millions of websites, is written in C.

8. Scientific & Engineering Applications

- Used for: Simulations, modeling, numerical computation
- These applications require **high performance**, **precision**, and **custom data handling**.
- C supports integration with mathematical libraries like **BLAS, LAPACK**, etc.
- Used in **climate modeling, molecular simulation, signal processing**, and more.
- **Example**:
  NASA's scientific computing applications have used C for performance-sensitive simulations.

Features of C language

- **Simplicity**: Few keywords and clean syntax.
- **Efficiency**: Closer to hardware, thus fast.
- **Portability**: Write on one machine, run on another with minor changes.
- **Modularity**: Code is organized in functions.
- **Rich Library**: Standard libraries support I/O, math, etc.
- **Pointer Support**: Direct access to memory.
- **Structured Programming**: Encourages code reuse and readability.

1. Simplicity: Few Keywords and Clean

What it means:

- C has a **small set of 32 keywords** (like if, while, for, int, return) which are easy to learn.
- Its **syntax is straightforward**, and programs follow a logical structure with minimal special symbols or confusing rules.

Why it matters:

- Beginners can focus more on **problem-solving and logic** than memorizing syntax.
- Helps in writing **concise and clean code**.

2. Closer to Hardware, Thus Fast

What it means:
C gives you **low-level access** to memory and hardware through pointers, bitwise operators, and direct register access (via inline assembly).
There is **no overhead** like in languages with garbage collection or runtime interpreters.
Why it matters:
C programs **run faster** than those written in high-level languages like Python or Java.
Ideal for **performance-critical systems** like OS kernels, games, or embedded software.

3. Portability: Write Once, Compile

What it means:

- C code can be compiled and run on **any hardware platform** with a C compiler.
- You may need to make **minor changes** (like header files or hardware-specific code), but the logic remains the same.

Why it matters:

- C is used to develop **cross-platform applications**.
- The same C program can run on Windows, Linux, or even embedded systems like Raspberry Pi or microcontrollers.

4. Modularity: Code is Organized in

What it means:

- C encourages breaking a large program into **smaller units called functions**.
- Each function handles a specific task, like input, processing, or output.

Why it matters:

- Makes code easier to **debug**, **read**, and **reuse**.
- Promotes **divide and conquer** approach in programming.

5. Rich Library: Standard Libraries Support

What it means:

- C provides built-in libraries for **input/output, string handling, memory allocation, math**, etc.
- These are available through **header files** like stdio.h, math.h, stdlib.h.

Why it matters:

- Saves time for the programmer—you don’t need to write everything from scratch.
- Increases productivity and functionality

6. Pointer Support: Direct Access to

What it means:

- C allows you to create **pointers**, which are variables that store memory addresses.
- This gives you **fine-grained control** over how data is accessed and stored.

Why it matters:

- Crucial for building **dynamic data structures** (linked lists, trees).
- Enables **efficient memory management**.
- Needed in **system-level programming** (e.g., memory-mapped I/O, device drivers).

7. Structured Programming: Encourages

What it means:

- C follows a structured, top-down design approach.
- It uses control structures (if, for, while, switch) and functions to create **logical blocks** of code.

Why it matters:

- Encourages students to write **clean, readable, and maintainable code**.
- Easy to follow the program flow from start to finish.
- Supports code reuse through **functions and modular design**.

Structure of C Program

```
#include <stdio.h>          // Preprocessor directive
int main() {                // Main function
printf("Hello World");      // Statement
 return 0;                  // Exit status
 }
```

Parts of C Program

- #include <stdio.h> → Includes standard I/O functions.
- Main() function → Entry point of every C program.
- Printf() function → Outputs data to the console.
- Return 0; → Exits the program, returning 0 to the OS.

Other parts in programs:

- Variable Declarations
- Statements & Expressions
- Comments (// or /* */)

Execution Flow of a C program

- **Writing**: Code is written in a (.c) file extension.
- **Preprocessing**: Preprocessor handles (#include), macros, etc..
- **Compilation**: Code is translated into assembly code.
- **Assembly**: Translates assembly into machine-level object code.
- **Linking**: Links code with libraries, creates executable.
- **Execution**: Executable is run on the system.

Reading a Character

```
char ch;

ch = getchar();  // Reads a single character from user input
```

- getchar() waits for the user to press a key and stores it in ch.
- Useful for reading individual characters or key presses.

Writing a Charater

```
char ch = 'A';
putchar(ch); // Outputs a single character
```

- putchar() prints a single character.
- Simple and fast way to output characters.

Formatted Input

- Taking input with formatting (e.g., numbers, text):

```
int a;
scanf("%d", &a);   // Reads formatted data (int here)
```

- scanf() reads user input.
- %d tells the compiler to expect an **integer**.
- &a means we're giving the **address of variable a** to store the input.

Formatted Output

- Displaying values with text:

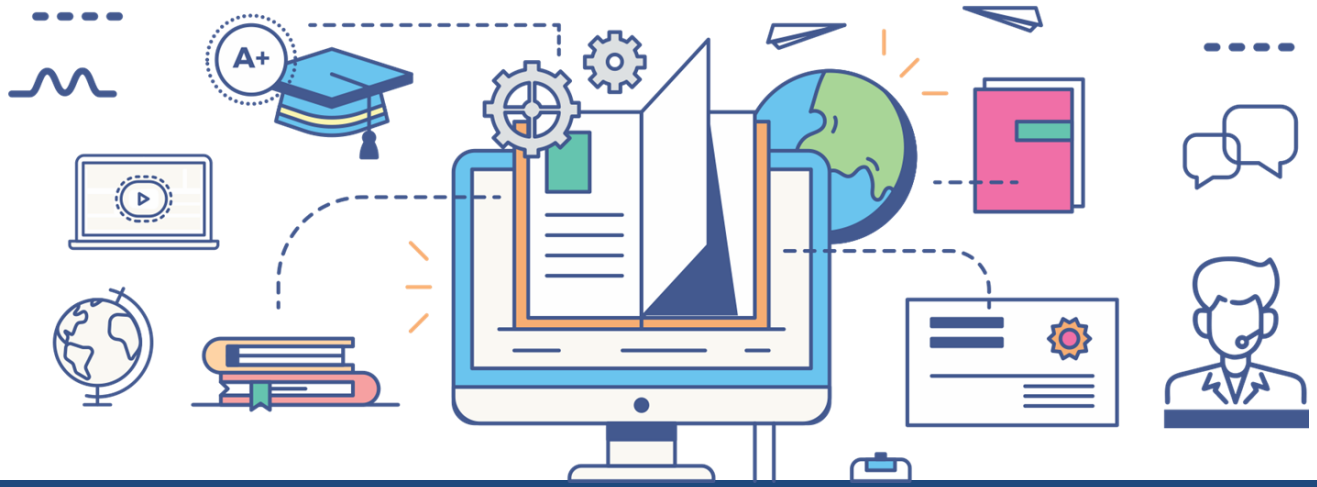
```
int a = 10;

printf("Value of a: %d", a);   // Prints formatted output
```

- printf() shows the value of ‘a’ along with text.
- %d will be replaced by the value of ‘a’ in the output.

Common Format Specifier

- %d → Integer
- %f → Float
- %c → Character
- %s → String

DIGITAL LEARNING CONTENT



Parul® University

www.paruluniversity.ac.in