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

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.

Parul®
University

DIGITAL LEARNING CONTENT

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.

Parul®
University

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**.

Parul® University

DIGITAL LEARNING CONTENT

C Editor

- **Syntax Highlighting:**
- Different elements of the C language (keywords, comments, strings) are displayed in distinct colors for improved readability.
- **Code Completion/IntelliSense:**
- Suggestions for variable names, function calls, and library functions appear as the user types, accelerating coding and reducing errors.
- **Error Highlighting:**
- Potential syntax errors or warnings are flagged directly within the editor, often with squiggly underlines or specific icons.
- **Code Formatting:**
- Tools to automatically indent code, align elements, and apply consistent styling.
- **Integrated Terminal/Output Window:**
- Allows for direct compilation and execution of C programs and viewing of program output.
- **Project Management:**
- Features to organize multiple source files, header files, and other project components.

C Debugger

- **Breakpoints:**
- Setting specific points in the code where program execution will pause, allowing inspection of variables and program state.
- **Step-by-Step Execution:**
- Executing the program line by line to observe the flow of control and variable changes. This includes:**Step Over:** Executes a function call as a single step without entering the function's code.
- **Step Into:** Enters the function's code when a function call is encountered.
- **Step Out:** Executes the remainder of the current function and stops at the line after the function call.
- **Watchpoints:**
- Setting conditions on variables or memory locations that will cause the program to pause when the condition is met (e.g., a variable's value changes).
- **Variable Inspection:**
- Viewing the current values of variables, arrays, and data structures during program execution.
- **Call Stack:**
- Examining the sequence of function calls that led to the current point of execution.

C Compiler

- **Preprocessing:** Handles preprocessor directives (e.g., `#include`, `#define`), expanding macros and including header files.
- **Compilation (Translation):** Converts the preprocessed source code into assembly language.
- **Assembly:** Translates the assembly code into machine code (object code).
- **Linking:** Combines the object code with necessary library functions and other object files to create a single executable program.

Key Aspects of C Compilation

Error Detection:
Compilers identify syntax errors, type mismatches, and other issues during the compilation phases.

Optimization:
Compilers can apply various optimizations to the generated code to improve performance and efficiency.

Portability:
C compilers allow source code to be compiled for different platforms, enabling the same C code to run on various operating systems and architectures.

Parul® University

DIGITAL LEARNING CONTENT

Structure of C Program

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

Structure of C Program

Documentation – comments about program

Link- instructions to compiler

Definition- defines all symbolic constants

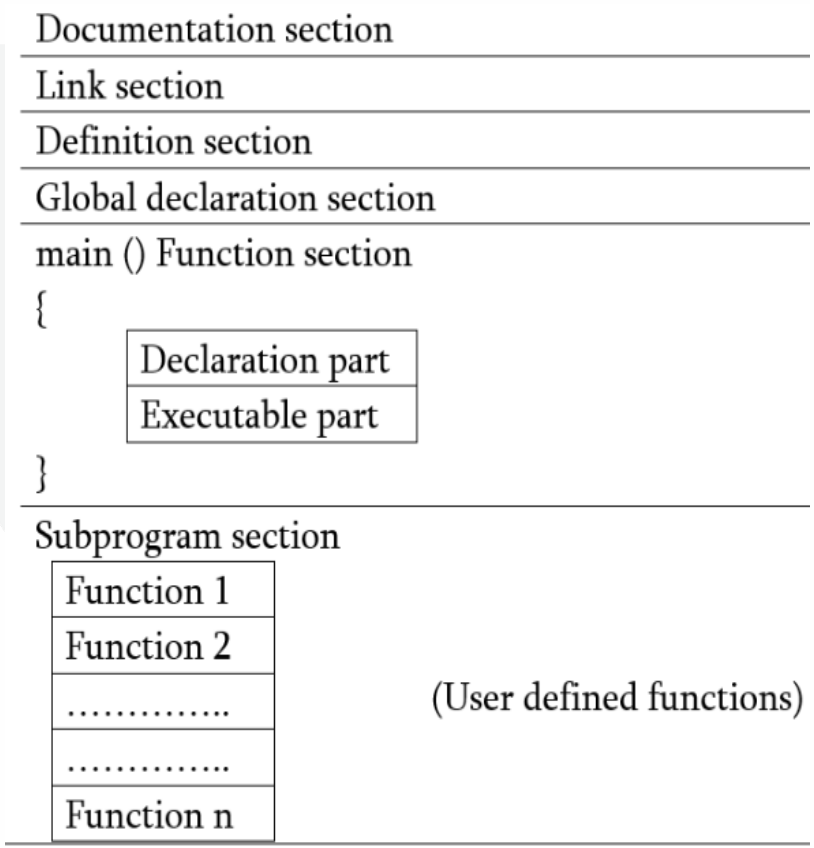
Global declaration - variable that is frequently used

Main()-

Declaration: all variables

Execution: programming

Subprogram- user defined functions



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.

Algorithm

- Programs=algorithms+data
- An algorithm is a part of computer program; an algorithm is an effective procedure for solving a problem in a finite number of steps.
- It is a complete step by step representation of the solution to a problem.

How to Design an Algorithm

- Step-1: Investigation
  - Identify process
  - Identify major decisions
  - Identify loops
  - Identify variables

Step-2: Preliminary algorithm

- Devise high level algorithm
- Walk-through algorithm. If any problem occurs then correct it.

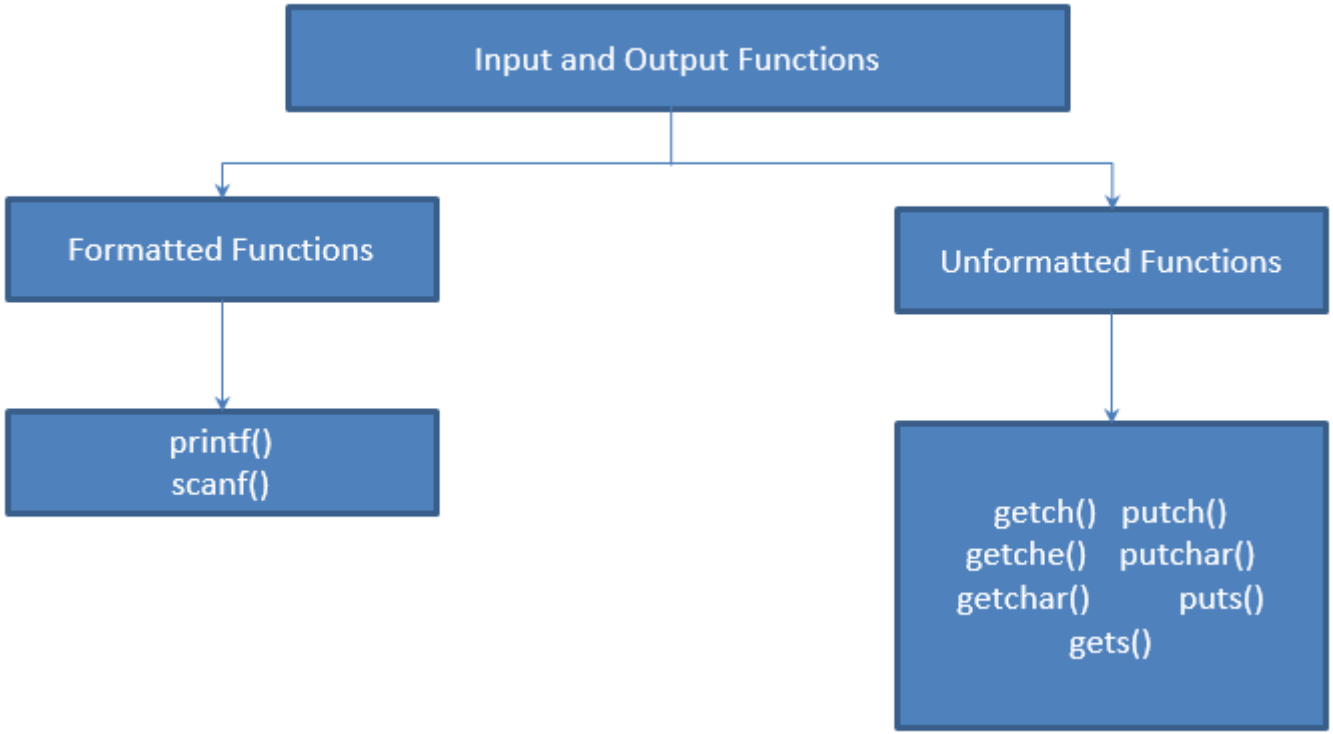
Step-3: Refining the algorithm

- Incorporate any refinements from step 2
- Group process
- Group variables
- Teat algorithm again

Example: Find sum of two numbers

1. Read value of a
2. Read value of b
3. Add a and b
4. Display the summation

Managing Input and Output Operations



Parul®
University

DIGITAL LEARNING CONTENT

Managing Input and Output Operations

Formatted Functions

- It read and write all types of data values.
- Require format string to produce formatted result
- Returns value after execution

Unformatted Functions

- Works only with character data type
- Do not require format conversion for formatting data type

Formatted function: scanf() function

- scanf() function reads all the types of data values.
- It is used for runtime assignment of variables.
- The scanf() statement also requires conversion symbol to identify the data to be read during the execution of the program.
- The scanf() stops functioning when some input entered does not match format string.

Formatted function: scanf() function

Syntax :

```
scanf("%d %f %c", &a, &b, &c);
```

- Scanf statement requires '&' operator called address operator
- The address operator prints the memory location of the variable
- scanf() statement the role of '&' operator is to indicate the memory location of the variable, so that the value read would be placed at that location.
- The scanf() function statement also return values. The return value is exactly equal to the number of values correctly read.
- If the read value is convertible to the given format, conversion is made.

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.

Examples

```
void main()
{
    int a;
    clrscr();
    printf(“Enter value of ‘A’ : “);
    scanf(“%c”, &a);
    printf(“A : %c”,a);
}
```

OUTPUT

Enter value of ‘A’ : 8

A : 8

Examples

```
void main()
{
    char a;
    clrscr();
    printf("Enter value of 'A' : ");
    scanf("%d", &a);
   printf("A : %d",a);
}
```

OUTPUT

Enter value of 'A' : 255

A : 255

Enter value of 'A' : 256

A : 256

Parul® University

DIGITAL LEARNING CONTENT

Formatted function: printf() function

1. This function displays output with specified format
2. It requires format conversion symbol or format string and variables names to the print the data
3. The list of variables are specified in the printf() statement
4. The values of the variables are printed as the sequence mentioned in printf()
5. The format string symbol and variable name should be the same in number and type

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.

Examples

```
void main()
{
    int NumInt = 2;
   float NumFloat=2.2;
   char LetterCh = 'C';

    printf("%d %f %c", NumInt, NumFloat, LetterCh);
}
```

Output :

2 2.2000 C

Examples

```
void main()
{
    int NumInt  = 65;
    clrscr();
    printf("%c %d", NumInt, NumInt);
}
```

Output :

A 65

Examples

```
void main()
{
    int NumInt  = 7;
    clrscr();
    printf("%f", NumInt);

}
```

Output :

Error Message : "Floating points formats not linked"

Format Specifiers

- All the format specification starts with % and a format specification letter after this symbol.
- It indicates the type of data and its format.
- If the format string does not match with the corresponding variable, the result will not be correct.
- Along with format specification use
  - Flags
  - Width
  - Precision

Flag and Width

Flag	It is used for output justification, numeric signs, decimal points, trailing zeros. The flag (-) justifies the result. If it is not given the default result is right justification.
Width	It sets the minimum field width for an output value. Width can be specified through a decimal point or using an asterisk '*'.

Common Format Specifier

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

Examples

```
void main()
{
    float g=123.456789;
    clrscr();
    printf("\n  %.1f", g);
    printf("\n  %.2f", g);
    printf("\n             %.3f", g);
    printf("\n  %.4f", g);
}
```

OUTPUT

123.5

123.46

123.457

123.4568

Format Specifier - Tables

Sr. No1	Format	Meaning	Explanation
1	%wd	Format for integer output	w is width in integer and d is conversion specification
2	%w.cf	Format for float numbers	w is width in integer, c specifies the number of digits after decimal point and f specifies the conversion specification
3	%w.cs	Format for string output	w is width for total characters, c are used displaying leading blanks and s specifies conversion specification

Format Specifier - Tables

Data Type		Format string
Integer	Short Integer	%d or %i
	Short unsigned	%u
	Long signed	%ld
	Long unsigned	%lu
	Unsigned hexadecimal	%u
	Unsigned octal	%o
Real	Floating	%f or %g
	Double Floating	%lf
Character	Signed Character	%c
	Unsigned Character	%c
	String	%s
Octal number		%o
Displays Hexa decimal number in lowercase		%hx
Displays Hexa decimal number in lowercase		%p
Aborts program with error		%n

Escape sequence

- printf() and scanf() statement follows the combination of characters called escape sequence
- Escape sequence are special characters starting with '\'

Escape Sequence	Use	ASCII value
\n	New Line	10
\b	Backspace	8
\f	Form feed	12
\'	Single quote	39
\\	Backslash	92
\0	Null	0
\t	Horizontal Tab	9
\r	Carriage Return	13
\a	Alert	7
\"	Double Quote	34
\v	Variable tab	11
\?	Question mark	63

Examples

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

void main()
{
   int a = 1, b = a + 1, c = b + 1, d = c + 1;
   //clrscr();

   printf("\tA = %d\nB = %d 'C = %d'", a, b, c);
   printf("\n\b*** D = %d **", d);
   printf("\n**********");
   printf("\nrA = %d B = %d", a, b);
    getch();
}
```

OUTPUT

A = 1
B = 2 'C = 3'
*** D = 4 **
****
A = 1 B = 2**

Unformatted functions

- C has three types of I/O functions
  - Character I/O
  - String I/O
  - File I/O

getchar()

- This function reads a character type data from standard input.
- It reads one character at a time till the user presses the enter key.
- Syntax

```
VariableName = getchar();
```

- Example

```
char c;
c = getchar();
```

putchar()

- This function prints one character on the screen at a time, read by the standard input.
- Syntax
  - putchar(variableName)
- Example

```
char c = 'C';
putchar(c);
```

getch() and getche()

- These functions read any alphanumeric character from the standard input device.
- The character entered is not displayed by the getch() function.
- The character entered is displayed by the getche() function.
- Exampe
  - ch = getch();
  - ch = getche();

gets()

- This function is used for accepting any string through `stdin` keyword until enter key is pressed.
- The header file `stdio.h` is needed for implementing the above function.
- Syntax

```
char str[length of string in number];
gets(str);
void main()
{
        char ch[30];
        clrscr();
        printf("Enter the string : ");
        gets();
        printf("\n Entered string : %s", ch);
}
```

puts()

- This function prints the string or character array.
- It is opposite to gets()

```
char str[length of string in number];
gets(str);
puts(str);
```

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.

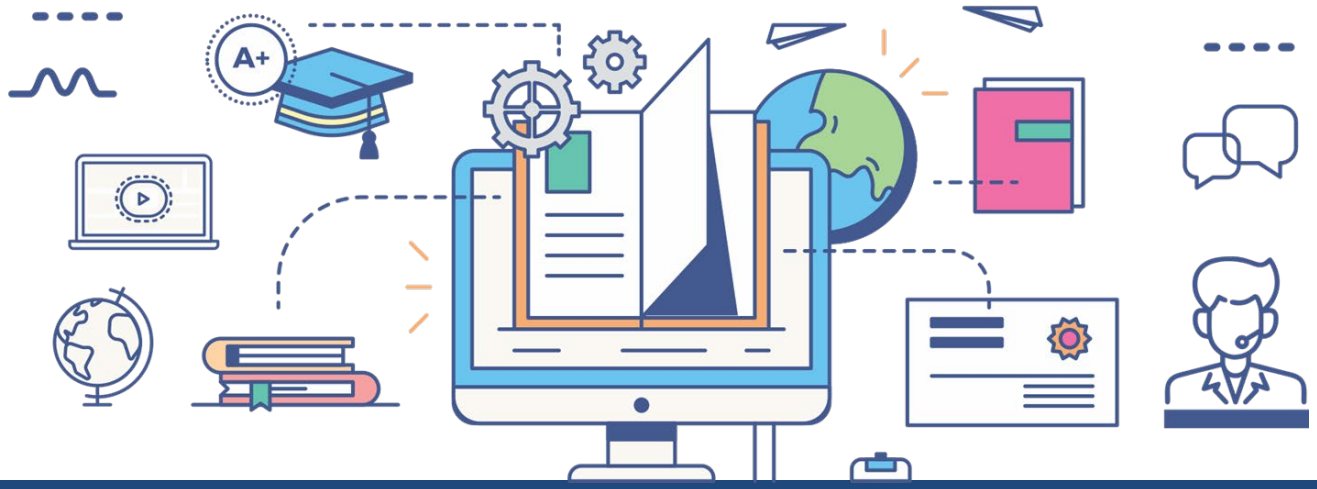
Writing a Charater

```
char ch = 'A';

putchar(ch); // Outputs a single character
```

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

DIGITAL LEARNING CONTENT



Parul® University

www.paruluniversity.ac.in