# MAA OMWATI DEGREE COLLEGE HASSANPUR

# (PALWAL)

## NOTES

## B.Com 4th SEM

## Structural Programming & Computer Graphics-II

# Unit-1

## 🔷 Introduction to C Language

### ✅ What is C?

C is a **general-purpose, procedural programming language** developed in the early 1970s by **Dennis Ritchie** at **Bell Labs**. It was initially designed to develop the **Unix operating system**, and it soon became the foundation of many modern programming languages such as **C++, Java, C#, and Objective-C**.

C is known for its **efficiency, portability, low-level memory access**, and its ability to be compiled and executed on a wide range of computer systems.

---

### ✅ Features of C

- **Simple and Powerful**: Though it has a small set of keywords, it is highly powerful.
- **Structured Language**: Encourages modular programming with functions.
- **Portable**: Programs written in C can be run on different machines with little or no modification.
- **Efficient**: Close to hardware, C provides low-level memory manipulation.
- **Extensible**: Allows defining new data types and functions.

---

## 🔷 Programming Rules in C

Before writing a C program, it's essential to understand its **syntax rules** and **structure**.

### ✳️ Basic Rules:

1. **Every statement ends with a semicolon (;)**.
2. **Curly braces ({})** are used to define blocks of code.
3. **C is case-sensitive** (Variable and variable are different).
4. A **C program must have a main() function**, which is the entry point.
5. **Comments** are written using // for single line or /* */ for multi-line.

### ✳️ Structure of a C Program:

c

CopyEdit

```
#include <stdio.h> // Preprocessor directive


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

---

### 🔷 Executing a C Program

To run a C program, follow these steps:

### ✅ 1. Write Code

Create a .c file using any text editor (Notepad, VS Code, etc.)

### ✅ 2. Compile

Use a **C compiler** like GCC to convert code into machine language.

bash

CopyEdit

gcc program.c -o program

### ✅ 3. Execute

Run the compiled executable.

bash

CopyEdit

./program

Most IDEs like Code::Blocks or Turbo C automate this process.

---

### 🔷 Operators in C

Operators are symbols used to **perform operations** on variables and values.

### ✅ Types of Operators:

#### 🔹 Arithmetic Operators

+, -, *, /, % (modulus)

#### 🔹 Relational Operators

==, !=, <, >, <=, >=

#### 🔹 Logical Operators

&& (AND), || (OR), ! (NOT)

#### 🔹 Assignment Operators

=, +=, -=, *=, /=

🔹 **Increment/Decrement**

++, --

🔹 **Bitwise Operators**

&, |, ^, ~, <<, >>

🔹 **Ternary Operator**

condition ? expression1 : expression2

---

🔷 **Decision Statements**

These allow a program to **make decisions** and **choose between different paths**.

✅ **1. if Statement**

c

CopyEdit

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

✅ **2. if-else Statement**

c

CopyEdit

```
if (condition) {
   // if block
} else {
   // else block
}
```

### ✅ 3. else if Ladder

c

CopyEdit

```
if (cond1) { }
else if (cond2) { }
else { }
```

### ✅ 4. switch Statement

Used for **multi-way branching** based on constant values.

c

CopyEdit

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

---

### 🔷 Loop Control Statements

These allow code to be **executed repeatedly**.

### ✅ 1. for Loop

c

CopyEdit

```
for (int i = 0; i < 5; i++) {
```

```
    printf("%d\n", i);
}
```

### ✅ 2. while Loop

```
c
CopyEdit
int i = 0;
while (i < 5) {
    printf("%d\n", i);
    i++;
}
```

### ✅ 3. do-while Loop

```
c
CopyEdit
int i = 0;
do {
    printf("%d\n", i);
    i++;
} while (i < 5);
```

### ✅ 4. break and continue

- break exits a loop early.
- continue skips the current iteration.

---

### 🔷 Structured Programming

Structured programming is a paradigm that promotes the use of:

- **Modular code** (functions)
- **Control structures** (if, loops)
- **Top-down design**

✅ **Advantages:**

- Easier to read and maintain.
- Enhances reusability.
- Reduces debugging and testing time.
- Promotes clarity and simplicity.

✅ **Disadvantages:**

- Not ideal for programs requiring asynchronous behavior.
- Sometimes less efficient than assembly for low-level tasks.
- Deep function nesting can affect readability.

---

🔷 **Input and Output in C**

✅ **printf() Function**

Used to **display output** on the screen.

✳️ **Syntax:**

c

CopyEdit

printf("format string", variable1, variable2, ...);

✳️ **Example:**

c

CopyEdit

int age = 25;

printf("Age is: %d", age);

**Format Specifiers:**

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

---

### ✅ scanf() Function

Used to **take input** from the user.

### ✳️ Syntax:

c

CopyEdit

scanf("format string", &variable1, &variable2, ...);

### ✳️ Example:

c

CopyEdit

int age;

scanf("%d", &age);

**Note:** The & (address-of operator) is essential for scanf() to store input in the memory location of the variable.

---

## 🔍 Example Program: Using scanf() and printf()

c

CopyEdit

```
#include <stdio.h>

int main() {
    int a, b, sum;
    printf("Enter two numbers: ");
    scanf("%d %d", &a, &b);
    sum = a + b;
    printf("Sum is: %d", sum);
    return 0;
}
```

# Unit-2

## ✅ 1. Introduction to Pointers

A **pointer** in C is a variable that stores the **memory address** of another variable. Pointers provide powerful capabilities such as dynamic memory allocation, efficient array handling, and the ability to simulate call-by-reference.

### ✳️ Why use Pointers?

- To access and manipulate memory locations directly.
- To pass large structures or arrays to functions efficiently.
- To allocate dynamic memory (with malloc, calloc, free).
- To build complex data structures like linked lists, trees, and graphs.

---

## ✅ 2. Pointer Declaration

### ✳️ Syntax:

c

CopyEdit

data_type *pointer_name;

### ✳️ Example:

c

CopyEdit

int a = 10;

int *p;

p = &a;

- *p is a pointer to an integer.
- &a returns the address of a.

🔹 **Important Symbols:**

- * (Asterisk): Declares a pointer or dereferences it.
- & (Ampersand): Fetches the address of a variable.

---

✅ **3. Operations on Pointers**

🔸 **Pointer Initialization:**

c

CopyEdit

int x = 5;

int *ptr = &x;

🔸 **Pointer Dereferencing:**

Accessing the value using a pointer.

c

CopyEdit

printf("%d", *ptr); // Output: 5

🔸 **Pointer Arithmetic:**

You can perform arithmetic operations on pointers:

- Increment (ptr++)
- Decrement (ptr--)
- Addition/Subtraction

c

CopyEdit

```
int arr[] = {10, 20, 30};
int *ptr = arr;
printf("%d", *(ptr + 1)); // Output: 20
```

---

### ✅ 4. Array of Pointers to Arrays

Pointers and arrays are closely related. You can create:

- Pointer to a single array element
- Array of pointers
- Pointer to an entire array

### ✳️ Example: Array of pointers

c

CopyEdit

```
int *arr[3];
int a = 10, b = 20, c = 30;
arr[0] = &a;
arr[1] = &b;
arr[2] = &c;

printf("%d", *arr[1]); // Output: 20
```

### ✳️ Pointer to an array

c

CopyEdit

```
int arr[5] = {1,2,3,4,5};
int (*ptr)[5] = &arr;
```

---

## 🔷 FUNCTIONS IN C

### ✅ 1. Function Definition

A **function** is a block of reusable code that performs a specific task.

#### ✳️ Syntax:

c

CopyEdit

```
return_type function_name(parameter_list) {
   // body
}
```

### ✅ 2. Function Prototype

Declares a function to the compiler before its actual use.

c

CopyEdit

```
int add(int, int); // prototype
```

### ✅ 3. Calling a Function

c

CopyEdit

```
int result = add(10, 20);
```

### ✅ 4. Passing Parameters to Functions

- **Call by Value:**

Copies the value; changes in the function do not affect the original.

c

CopyEdit

```
void fun(int x) {
   x = x + 10;
}
```

🔹 **Call by Reference (via Pointers):**

c

CopyEdit

```
void fun(int *x) {
   *x = *x + 10;
}
```

---

✅ **5. Recursion**

A **recursive function** is a function that calls itself.

✳️ **Example:**

c

CopyEdit

```
int factorial(int n) {
   if(n == 0) return 1;
   else return n * factorial(n - 1);
}
```

---

## 🔷 DATA STRUCTURES IN C

### ✅ 1. Array

An **array** is a collection of elements of the same type stored in contiguous memory.

#### ✳️ Syntax:

c

CopyEdit

int arr[5] = {1,2,3,4,5};

#### 🔹 Access:

c

CopyEdit

printf("%d", arr[2]); // Output: 3

Arrays can be:

- One-dimensional
- Multi-dimensional

---

### ✅ 2. Overview of Compilers and Interpreters

#### 🔹 Compiler:

A **compiler** translates the entire source code into machine code **before execution**.

#### 🔹 Interpreter:

An **interpreter** translates and executes the code **line by line**.

---

### ✅ 3. Program Development in C

✳️ **Steps:**

1. **Edit**: Write the source code in .c file.
2. **Preprocess**: Includes header files and macro expansions.
3. **Compile**: Convert to assembly or intermediate code.
4. **Assemble**: Generate object code.
5. **Link**: Resolve external references and generate executable.
6. **Load and Execute**

---

✅ **4. Difference between Compiler and Interpreter**

| **Feature** | **Compiler** | **Interpreter** |
|---|---|---|
| Translation | Whole code at once | One line at a time |
| Speed | Faster execution after compilation | Slower |
| Error Reporting | After entire code | Line by line |
| Output | Machine code (executable) | No separate output |
| Example Languages | C, C++, Java (partly) | Python, JavaScript, BASIC |

# Unit-3

## 🔷 1. Data Handling: Formatted and Unformatted Console Functions in C

In C, **console input/output functions** are used for data handling — reading from or writing to the console. These can be broadly divided into:

### ✅ A. Formatted I/O Functions

Formatted I/O provides a structured and specific way to read and write data using format specifiers.

#### 🔹 printf() - Formatted Output

Used to display output in a formatted way.

c

CopyEdit

int a = 10;

printf("The value is %d", a);

- %d → for integer
- %f → for float
- %c → for character
- %s → for string

#### 🔹 scanf() - Formatted Input

Used to read formatted data from the console.

c

CopyEdit

int age;

scanf("%d", &age);

- 🔹 **Key Features:**
  - Supports multiple data types.
  - Allows formatted precision.
  - Requires **format specifiers**.

✅ **B. Unformatted I/O Functions**

These are simpler and do not use format specifiers.

- 🔹 **getchar() and putchar() – character I/O**

c

CopyEdit

char ch = getchar(); // Reads a single character

putchar(ch); // Prints a single character

- 🔹 **gets() and puts() – string I/O (Note: gets() is unsafe and deprecated)**

c

CopyEdit

char name[20];

gets(name); // Reads a string including whitespaces

puts(name); // Prints the string

- 🔹 **Differences:**

| Feature | Formatted I/O (printf, scanf) | Unformatted I/O (getchar, puts) |
|---|---|---|
| Precision Control | Yes | No |
| Format Specifier | Required | Not used |
| User Control | More precise | Less control |
| Safety | Safer (with format specifiers) | gets() is unsafe |

---

### 🔷 2. Difference Between Structures and Unions

Both structures and unions are **user-defined data types** that group different types of variables under one name. However, they differ in **memory allocation and data handling**.

#### ✅ A. Structure

- All members have **separate memory locations**.
- Can store **multiple values simultaneously**.

c

CopyEdit

```
struct Person {
  int age;
  float weight;
};
```

#### ✅ B. Union

- All members **share the same memory location**.

- Can store **only one value at a time**.

c

CopyEdit

```
union Data {
  int i;
  float f;
};
```

🔹 **Differences:**

| Feature | Structure | Union |
|---|---|---|
| Memory | Allocated for all members | Shared among all members |
| Size | Sum of all members | Size of largest member |
| Usage | Use when multiple members needed | Use when one member used at a time |
| Value storage | All at once | One at a time |

---

### 🔷 3. Nesting of Structures

**Nested structures** allow one structure to be a member of another structure, making it useful for modeling complex data.

✅ **Example:**

c

CopyEdit

```
struct Date {
  int day, month, year;
```

};

struct Employee {
  int id;
  char name[20];
  struct Date joiningDate;
};

✅ **Accessing Nested Members:**

c

CopyEdit

struct Employee emp;

emp.joiningDate.day = 10;

**Applications:**

- Employee Records
- Student Information Systems
- Billing Systems

---

🔷 **4. Searching: Sequential and Binary Searching in Arrays**

Searching is the process of finding a specific element in a data set.

---

✅ **A. Sequential (Linear) Search**

Checks every element until the target is found.

🔹 **Algorithm:**

1. Start from the first element.

2. Compare with the target.
3. If match found, return index.
4. Else, continue till end.

🔹 **Code:**

c

CopyEdit

```
int search(int arr[], int n, int target) {
   for (int i = 0; i < n; i++)
      if (arr[i] == target)
         return i;
   return -1;
}
```

🔹 **Time Complexity:**

- **Best Case:** O(1)
- **Worst Case:** O(n)

---

✅ **B. Binary Search**

Efficient algorithm that works on **sorted arrays** by dividing the array in half repeatedly.

🔹 **Algorithm:**

1. Find middle element.
2. If match → done.
3. If target < mid → search left.
4. If target > mid → search right.

- **Code:**

c

CopyEdit

```
int binarySearch(int arr[], int low, int high, int key) {
    while (low <= high) {
        int mid = (low + high)/2;
        if (arr[mid] == key)
            return mid;
        else if (key < arr[mid])
            high = mid - 1;
        else
            low = mid + 1;
    }
    return -1;
}
```

- **Time Complexity:**
  - **Best Case:** O(1)
  - **Worst Case:** O(log n)

---

- **Comparison:**

| Feature | Sequential Search | Binary Search |
| --- | --- | --- |
| Data Requirement | Unsorted/Sorted | Sorted only |
| Efficiency | Slower (O(n)) | Faster (O(log n)) |

| Feature | Sequential Search | Binary Search |
| --- | --- | --- |
| Complexity | Simple | Complex |

---

### 🔷 5. Difference Between While, Do-While, and For Loop

All three loops are **control flow statements** used for iteration in C, but they differ in syntax and behavior.

### ✅ A. While Loop

Executes **as long as condition is true**. Condition is **checked first**.

c

CopyEdit

```
int i = 0;
while (i < 5) {
    printf("%d", i);
    i++;
}
```

### ✅ B. Do-While Loop

Executes **at least once**, then continues if condition is true. Condition is checked **after** execution.

c

CopyEdit

```
int i = 0;
do {
    printf("%d", i);
    i++;
```

} while (i < 5);

### ✅ C. For Loop

Used when the number of iterations is known. Combines initialization, condition, and update in a single line.

c

CopyEdit

```
for (int i = 0; i < 5; i++) {
    printf("%d", i);
}
```

🔹 **Comparison:**

| Feature | while | do-while | for |
|---|---|---|---|
| Entry/Exit check | Entry check | Exit check | Entry check |
| Executes once? | No | Yes (always at least once) | No |
| Best for | Unknown iteration | Menu-driven / At-least-once | Known iteration count |
| Syntax | Separate steps | Separate steps | All-in-one |

# Unit-4

## 🔷 What is Computer Graphics?

**Computer Graphics (CG)** is the field of visual computing where one utilizes computers both to generate visual images synthetically and to integrate or alter visual and spatial information sampled from the real world. In simpler terms, it involves creating, manipulating, and representing visual content using computational methods.

---

## 🔷 Computer Graphics Applications

Computer Graphics has a wide variety of practical and theoretical applications in modern technology. These applications impact nearly every domain:

### ✅ 1. Computer-Aided Design (CAD)

CAD involves using computer graphics to create, modify, analyze, and optimize designs.

- **Industries Involved**: Automotive, aerospace, architecture, and electronics.
- **Benefits**:
  - High precision in engineering drawings.
  - Easy testing and simulation.
  - Time and cost-efficient prototyping.
- **Tools**: AutoCAD, SolidWorks, CATIA.

### ✅ 2. Presentation Graphics

Used to visually represent information, data, or messages through charts, diagrams, graphs, etc.

- **Examples**:
  - PowerPoint presentations.
  - Infographics and dashboards.
  - Business and educational reports.
- **Purpose**:
  - Communicate ideas effectively.
  - Support storytelling in data presentation.

### ✅ 3. Computer Art

Artists use computer graphics as a medium of expression and creativity.

- **Forms**:
  - Digital painting
  - Fractal art
  - Generative art
- **Software Tools**:
  - Adobe Illustrator
  - CorelDRAW
  - Blender (for 3D art)

### ✅ 4. Entertainment

Used extensively in movies, video games, and virtual reality environments.

- **Fields**:
  - 2D and 3D animation

    - CGI (Computer-Generated Imagery)
    - Game design and cinematics
- **Technologies**:
    - Maya, Unity, Unreal Engine

### ✅ 5. Education and Training

Enhances learning and training through interactive and immersive environments.

- **Uses**:
    - Simulations (flight, driving)
    - Virtual labs
    - E-learning modules

### ✅ 6. Visualization

Turning complex data into understandable and visually appealing formats.

- **Examples**:
    - Scientific visualization (e.g., weather models)
    - Medical imaging (MRI, CT scans)
    - Financial data visualization

### ✅ 7. Image Processing

Involves processing images to enhance quality or extract meaningful information.

- **Applications**:
    - Image restoration
    - Object recognition

- Face detection
- Medical diagnostics

### ✅ 8. Graphical User Interfaces (GUIs)

GUIs allow users to interact with computers through graphical elements like windows, icons, and buttons.

- **Benefits**:
  - Ease of use
  - Intuitive controls
  - Rich user experience
- **Examples**:
  - Operating systems (Windows, macOS)
  - Mobile apps
  - Web-based interfaces

---

## 🔷 Display Devices in Computer Graphics

Display devices are essential hardware components that present graphics visually.

### ✅ 1. Cathode Ray Tube (CRT)

- Traditional monitor using electron beams.
- Works on raster scan or vector scan methods.
- Bulky but capable of high resolution and color depth.

### ✅ 2. Liquid Crystal Display (LCD)

- Flat-panel display using liquid crystals and backlight.
- Energy-efficient and lighter than CRT.

- Used in laptops, TVs, mobile devices.

✅ **3. LED and OLED Displays**

- More advanced than LCDs.
- OLED offers deep blacks and high contrast.
- Widely used in smartphones, smart TVs.

✅ **4. Plasma Displays, Projectors, and VR Headsets**

- Plasma: High contrast ratio.
- Projectors: Used for large displays.
- VR headsets: For immersive environments.

---

🔷 **Overview of Display Methods**

Display methods define how graphics are rendered on screen. There are primarily two types:

✅ **1. Raster Scan Displays**

- Most common method (used in CRT, LCD).
- Image is drawn pixel by pixel in horizontal lines.
- Each frame is refreshed at a certain rate (e.g., 60 Hz).

✅ **2. Random Scan (Vector Scan) Displays**

- Electron beam directly draws lines between coordinates.
- Suitable for line-art images like graphs.
- Used in early computer graphics systems.

---

🔷 **Raster Scan Display Processing Unit**

A **Raster Scan System** includes the following components:

✅ **Components:**

1. **Frame Buffer**: Memory that stores pixel color values.
2. **Display Controller**: Reads frame buffer and sends data to monitor.
3. **RAMDAC (Digital to Analog Converter)**: Converts digital pixel data to analog signals.
4. **CPU and Graphics Card**: Handles rendering of graphics.

✅ **Working:**

- The image is stored as a matrix of pixels in the frame buffer.
- The display controller reads pixel by pixel, row by row.
- Data is sent to the display through RAMDAC.

---

🔷 **Input Devices for Interactive Graphics**

Input devices allow user interaction with the graphics system.

✅ **1. Keyboard and Mouse**

- Standard devices for text and pointer inputs.

✅ **2. Trackballs and Joysticks**

- Common in gaming and control systems.

✅ **3. Light Pen**

- Used to draw directly on CRT displays.

✅ **4. Touch Screens and Stylus Pens**

- Found in modern devices like tablets, kiosks.

✅ **5. Graphics Tablets**

- Used by digital artists for precise drawing.

### ✅ 6. Scanners and Cameras

- Capture images and convert them into digital form for processing.

---

### 🔷 Programmer's Model of Interactive Graphics Systems

To build interactive graphics applications, programmers use a layered architecture that abstracts hardware complexities.

### ✅ Key Layers:

1. **Application Layer**:
    - The high-level software (e.g., a drawing program).
2. **Graphics Libraries**:
    - APIs like OpenGL, DirectX, WebGL provide functions for drawing.
3. **Device Driver Layer**:
    - Manages communication between the software and hardware.
4. **Hardware Layer**:
    - Physical components like GPU, display device, input device.

### ✅ Event Handling:

- Input devices generate events (e.g., mouse click).
- Event queue captures these and passes them to the application.

---

### 🔷 Storage Formats for Pictures

Image storage formats define how graphical data is stored and compressed.

✅ **1. Bitmap (BMP)**

- Stores raw pixel data.
- Uncompressed → large file size.
- Used in raster scan systems.

✅ **2. JPEG (Joint Photographic Experts Group)**

- Compressed format for photos.
- Lossy compression → reduced file size with slight quality loss.

✅ **3. PNG (Portable Network Graphics)**

- Lossless compression.
- Supports transparency.
- Ideal for web graphics and icons.

✅ **4. GIF (Graphics Interchange Format)**

- Limited to 256 colors.
- Supports simple animation.
- Used for web memes and banners.

✅ **5. TIFF (Tagged Image File Format)**

- High-quality images.
- Used in medical and printing fields.

✅ **6. SVG (Scalable Vector Graphics)**

- XML-based format.
- Stores vector data (lines, shapes).
- Scalable without quality loss.