Indian Institute of Management and Commerce
Course: B.Com(Computer Applications)
Subject: Programming with C & C++
Important Questions with Answers
Unit I

1) Explain the structure of a C program?

A) A **C program** follows a well-defined , every C program is built from a few key sections that work together to make the program run .

1. Documentation Section

This is optional and usually placed at the top of the program.

- Contains comments explaining the purpose of the program.
- Written using /* ... */ or //.

```
/* This program prints Hello, World */
```

2. Preprocessor Directives

These are instructions given to the compiler before the actual compilation starts.

- Begin with #
- Used to include libraries or define constants.

```
#include <stdio.h>
#define PI 3.14
```

- #include → includes standard libraries
- #define → defines constants or macros

3. Global Declarations

Variables or functions declared outside all functions.

- Accessible throughout the program.
- Used when multiple functions need the same data.

```
int a = 10;
```

4. main() Function

This is the **entry point** of every C program—execution starts here.

```
int main() {
   // code
   return 0;
}
```

- Every C program must have exactly one main() function.
- return 0; indicates successful execution.

5. Local Declarations

Variables declared inside functions.

```
int main() {
   int a = 5;  // local variable
}
```

- These variables are only accessible within that function.

6. Statements and Expressions

This is where the actual logic of the program is written.

- Includes input/output, calculations, conditions, loops, etc.

```
printf("Hello, World!");
```

7. User-Defined Functions

You can define your own functions to organize code and reuse logic.

```
void greet() {
   printf("Welcome!");
}
```

Example Program

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

// Global declaration
int num = 10;
```

```
// Function declaration
void greet();

int main() {         // Main function
   greet();         // Function call
   printf("Number = %d", num);
   return 0;
}

// Function definition
void greet() {
   printf("Hello, World!\n");
}
```

2. What are datatypes? explain different types of datatypes with syntax and example?

A) Data Types in C

Data types specify the **type of data** a variable can store in a program.

They help the compiler understand:

- how much memory to allocate
- what kind of value (integer, character, decimal, etc.)
- what operations are allowed.

Types of Data Types in C

1. Basic (Primary) Data Types

These are the fundamental data types.

(a) int

- Stores whole numbers

Syntax:

```
int variable_name;
```

Example:

```
int a = 10;
```

(b) float

- Stores decimal (single precision)

Syntax:

```
float variable_name;
```

Example:

```
float b = 3.14;
```

(c) double

- Stores large decimal numbers (double precision)

Syntax:

```
double variable_name;
```

Example:

```
double c = 12.345678;
```

(d) char

- Stores a single character

Syntax:

```
char variable_name;
```

Example:

```
char ch = 'A';
```

2. Derived Data Types

Created from basic data types.

(a) Array

- Collection of same type elements

Syntax:

```
data_type array_name[size];
```

Example:

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

(b) Pointer

- Stores address of another variable

Syntax:

```
data_type *pointer_name;
```

Example:

```
int a = 10;
int *p = &a;
```

(c) Function

- Block of code that performs a task

Syntax:

```
return_type function_name(parameters);
```

Example:

```
int add(int x, int y) {
   return x + y;
}
```

3. User-Defined Data Types

Created by the programmer.

(a) Structure (struct)

- Groups different data types

Syntax:

```
struct structure_name {
   data_type member1;
   data_type member2;
};
```

Example:

```
struct Student {
   int id;
   char name[20];
};
```

(b) Union (union)

- All members share the same memory

Syntax:

```
union union_name {
   data_type member1;
   data_type member2;
};
```

Example:

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

(c) Enumeration (enum)

- Assigns names to integer constants

Syntax:

```
enum enum_name {value1, value2, value3};
```

Example:

```
enum Day {Mon, Tue, Wed};
```

4. Void Data Type

- Represents no value.

(a) Void Function

Syntax:

```
void function_name();
```

Example:

```
void display() {
   printf("Hello");
}
```

3) what is an operator in C? explain types of operators?

Operator in C

An **operator** in C is a **symbol that performs an operation** on one or more operands (values or variables).

Example:

```
int a = 10, b = 5;
int c = a + b;   // '+' is an operator
```

Here, + adds two values.

Types of Operators in C

C provides several types of operators:

1. Arithmetic Operators

Used to perform mathematical calculations.

Operator	Meaning	Example
+	Addition	a + b
-	Subtraction	a - b
*	Multiplication	a * b
/	Division	a / b
%	Modulus (remainder)	a % b

Example:

```
int a = 10, b = 3;
printf("%d", a % b);  // Output: 1
```

2. Relational Operators

Used to compare two values (result is true/false).

Operator	Meaning
==	Equal to
!=	Not equal
>	Greater than
<	Less than
>=	Greater than or equal
<=	Less than or equal

Example:

```
if (a > b) {
   printf("a is greater");
}
```

3. Logical Operators

Used to combine conditions.

Operator	Meaning
&&	Logical AND
||	Logical OR
!	Logical NOT

Example:

```
if (a > 0 && b > 0) {
   printf("Both are positive");
}
```

4. Assignment Operators

Used to assign values to variables.

Operator	Example	Meaning
=	a = 5	Assign
+=	a += 2	a = a + 2

Operator	Example	Meaning
-=	a -= 2	a = a - 2
*=	a *= 2	a = a * 2
/=	a /= 2	a = a / 2
%=	a %= 2	a = a % 2

5. Increment and Decrement Operators

Operator	Meaning
++	Increment (add 1)
--	Decrement (subtract 1)

Example:

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

6. Bitwise Operators

Operate on binary values.

Operator	Meaning
&	AND
|	OR
^	XOR
~	NOT
<<	Left shift
>>	Right shift

Example:

```
int a = 5, b = 3;
printf("%d", a & b);  // Output: 1
```

7. Conditional (Ternary) Operator

- Works as a shortcut for if-else

Syntax:

```
condition ? expression1 : expression2;
```

Example:

```
int max = (a > b) ? a : b;
```

8. Special Operators

Some additional operators in C:

Operator	Purpose
sizeof	Returns size of variable
&	Address of variable
*	Pointer (indirection)
,	Comma operator

Example:

```
printf("%lu", sizeof(int));
```

Unit II

1) What are Conditional statements? Explain different types of conditional statements?

A) Conditional statements are used to make decisions in a program. They execute different blocks of code based on whether a condition is true or false.

Types of Conditional Statements in C

1. if Statement

Executes a block of code only if the condition is true.

Syntax:

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

Example:

```
int a = 10;
if (a > 5) {
   printf("a is greater than 5");
}
```

2. if-else Statement

Executes one block if condition is true, otherwise another block.

Syntax:

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

Example:

```
int a = 3;
if (a % 2 == 0) {
   printf("Even");
} else {
   printf("Odd");
}
```

3. else-if Ladder

Used when there are multiple conditions.

Syntax:

```
if (condition1) {
   // block1
} else if (condition2) {
   // block2
} else {
   // default block
}
```

Example:

```
int marks = 75;

if (marks >= 90) {
   printf("Grade A");
} else if (marks >= 60) {
   printf("Grade B");
} else {
   printf("Grade C");
}
```

4. Nested if

An if statement inside another if.

Example:

```
int a = 10, b = 5;

if (a > 0) {
```

```
    if (b > 0) {
        printf("Both are positive");
    }
}
```

5. switch Statement

Used when a variable is compared with multiple constant values.

Syntax:

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

Example:

```
void main()
{
int day = 2;

switch(day) {
    case 1: printf("Monday"); break;
    case 2: printf("Tuesday"); break;
    default: printf("Invalid");
   }
}
```

6. Conditional (Ternary) Operator

Short form of if-else.

Syntax:

```
condition ? expression1 : expression2;
```

Example:

```
int a = 10, b = 20;
int max = (a > b) ? a : b;
```

2) What is loop? Explain Looping statements.

Types of Looping Statements in C

C provides **three main types of loops**:

1. while Loop

- Executes the loop **as long as the condition is true**
- Condition is checked **before execution** (entry-controlled loop)

Syntax:

```
while (condition) {
   // statements
}
```

Example:

```
#include <stdio.h>
int main() {
   int i = 1;
   while (i <= 5) {
      printf("%d ", i);
      i++;
   }
   return 0;
}
```

Output: 1 2 3 4 5

2. do-while Loop

- Executes the loop **at least once**, even if the condition is false
- Condition is checked **after execution** (exit-controlled loop)

Syntax:

```
do {
   // statements
} while (condition);
```

Example:

```
#include <stdio.h>
int main() {
   int i = 1;
   do {
      printf("%d ", i);
      i++;
   } while (i <= 5);
   return 0;
}
```

Output: 1 2 3 4 5

3. for Loop

- Used when the number of iterations is known
- Combines initialization, condition, and increment in one line

Syntax:

```
for (initialization; condition; increment/decrement) {
   // statements
}
```

Example:

```
#include <stdio.h>
int main() {
   for (int i = 1; i <= 5; i++) {
      printf("%d ", i);
   }
   return 0;
}
```

Output: 1 2 3 4 5

Unit III

1) What is an Array? how to declare an array, Initialize an array with syntax and example?

A) Array in C:

An array is a collection of elements of the same data type stored in continuous memory locations and accessed using a single name with an index.

1. Declaration of an Array

Syntax:

data_type array_name[size];

Example:

int a[5];

This declares an array named **a** that can store 5 integers.

2. Initialization of an Array

(a) Initialization at Declaration

Syntax:

data_type array_name[size] = {values};

Example:

int a [5] = {10, 20, 30, 40, 50};

3. Accessing Array Elements

Array index starts from 0

Example:

```
printf("%d", a[0]);  // Access first element
```

Example Program

```
#include <stdio.h>
int main() {
int a[5] = {10, 20, 30, 40, 50};

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

3) What is a function ? Explain different types of functions?

A) A **function** is a **block of code** that performs a specific task and can be reused whenever needed.

It helps in:

- reducing code repetition
- improving readability
- dividing a program into smaller parts (modularity)

Basic Syntax of a Function

```
return_type function_name(parameters) {
   // statements
   return value;
}
```

Types of Functions in C

Functions in C are mainly classified into **two categories**:

1. Library Functions (Built-in Functions)
These are **predefined functions** provided by C libraries.

Examples:

- printf() – display output
- scanf() – take input
- sqrt() – calculate square root

Example:

```
#include <stdio.h>
int main() {
   printf("Hello");
   return 0;
}
```

2. User-Defined Functions
Functions created by the programmer.

Types of User-Defined Functions

Based on arguments and return values:

(a) Function with no arguments and no return value

Syntax:

```
void function_name() {
   // statements
}
```

Example:

```
void greet() {
   printf("Hello");
}
```

(b) Function with arguments and no return value

Syntax:

```
void function_name(parameters) {
   // statements
}
```

Example:

```
void add(int a, int b) {
   printf("%d", a + b);
}
```

(c) Function with arguments and return value

Syntax:

```
return_type function_name(parameters) {
   return value;
}
```

Example:

```
int add(int a, int b) {
   return a + b;
}
```

(d) Function with no arguments but return value

Syntax:

```
return_type function_name() {
   return value;
}
```

Example:

```
int getNumber() {
   return 10;
}
```

Example Program Using Function

```
#include <stdio.h>
int add(int x, int y)
{
   return x + y;
}

int main() {
   int result = add(5, 3);
   printf("%d", result);
   return 0;
}
```

4) Explain call by value and call by reference with example?

A) Call by Value and Call by Reference in C

These are two ways of passing arguments to functions.

1. Call by Value

Definition

In call by value, a copy of the actual value is passed to the function.
Changes made inside the function do not affect the original variable.

Example:

```
#include <stdio.h>
void change(int x) {
   x = x + 10;
   printf("Inside function: %d\n", x);
}
int main() {
   int a = 5;
   change(a);
   printf("Outside function: %d", a);
   return 0;
}
```

Output:
Inside function: 15
Outside function: 5

The value of a is not changed because only a copy is passed.

2. Call by Reference
Definition:
In call by reference, the address of the variable is passed to the function using pointers. Changes made inside the function affect the original variable.
Example:

```
#include <stdio.h>
void change(int *x) {
    *x = *x + 10;
    printf("Inside function: %d\n", *x);
}

int main() {
    int a = 5;
    change(&a);
    printf("Outside function: %d", a);
    return 0;
}
```

Output:
Inside function: 15
Outside function: 15

5) what is a string? explain string handling functions with syntax and example?

A **string** in C is a **collection of characters** stored in an array and terminated by a special character `\0` **(null character)**.

Example:

```
char str[] = "Hello";
```

Here, internally it is stored as: `H e l l o \0`
String Handling Functions

C provides several built-in string functions in the header file:

#include <string.h>

Common String Handling Functions

Function	**Syntax**	**Description**	**Example**
strlen()	strlen(str)	Returns the number of characters in a string (excluding the null terminator \0).	strlen("Hello") → 5
strcpy()	strcpy(dest, src)	Copies the source string into the destination array.	strcpy(d, "Hi") → d becomes "Hi"

strcat()	strcat(dest, src)	Appends (concatenates) the source string to the end of the destination string.	strcat("Hi ", "All") → "Hi All"
strcmp()	strcmp(s1, s2)	Compares two strings character by character; returns 0 if equal.	strcmp("A", "A") → 0
strrev()	strrev(str)	Reverses the characters in a string (Note: non-standard in some compilers).	strrev("ABC") → "CBA"
strupr()	strupr(s1,s2)	Converts all characters in a string to uppercase.	strlwr("hi ") → " HI "
strlwr()	strlwr(str)	Converts all characters in a string to lowercase.	strlwr("HI") → "hi"

6) Explain call by value and call by reference with example.

In **call by value**, a **copy of the actual value** is passed to the function.

Changes made inside the function **do NOT affect** the original variable.

Syntax Example

```
void function_name(data_type variable);
```

Example

```
#include <stdio.h>
void swap(int a, int b) {
    int temp;
    temp = a;
    a = b;
    b = temp;
}

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

Output

x = 10, y = 20

Even though swap is done inside the function, original values are unchanged because only copies were used.

2. Call by Reference

In **call by reference**, the **address of variables** is passed using pointers.

Changes inside the function **affect original variables**.

Syntax Example

```
void function_name(data_type *variable);
```

Example

```
#include <stdio.h>
void swap(int *a, int *b) {
    int temp;
    temp = *a;
    *a = *b;
    *b = temp;
}
int main() {
    int x = 10, y = 20;
    swap(&x, &y);
    printf("x = %d, y = %d\n", x, y);
    return 0;
}
```

Output

x = 20, y = 10

Addresses are passed, so the function directly modifies original values.

Unit IV

1) What is a pointer? Explain how to declare and access a pointer with syntax and example?

A **pointer** is a variable that stores the **memory address** of another variable instead of storing the actual value.

Declaration of a Pointer

Syntax:

```
data_type *pointer_name;
```

Example:

```
int *p;
```

Here, p is a pointer that can store the address of an integer variable.

Initialization (Assigning Address)

Use the **address operator &** to assign the address of a variable to a pointer.

```
int a = 10;
int *p;
p = &a;
```

Accessing a Pointer

Use the **dereference operator *** to access the value stored at the address.

```
printf("%d", *p);
```

Example

```
#include <stdio.h>
int main() {
    int a = 10;
    int *p;
    p = &a;  // storing address of 'a' in pointer
    printf("Value of a: %d\n", a);
    printf("Address of a: %p\n", &a);
    printf("Pointer p stores address: %p\n", p);
    printf("Value at address stored in p: %d\n", *p);
    return 0;
}
```

Output

- a = 10 → actual value
- &a → address of variable a

- p → stores address of a
- *p → gives value stored at that address (10)

2) Explain Structure with syntax and example?

A **structure** in C is a **user-defined data type** that allows you to group different types of variables under one name. It helps in organizing related data.

Syntax of Structure

```
struct structure_name {
   data_type member1;
   data_type member2;
   data_type member3;
};
```

Accessing Structure Members

Use the **dot operator (.)**

s1.id
s1.name
s1.marks

Example (Declaration & Initialization)

```
#include <stdio.h>

// structure definition
struct Student {
   int id;
   char name[20];
   float marks;
};

int main()
{
   // declaring structure variable
   struct Student s1 = {101, "Ravi", 85.5};
   // accessing structure members
   printf("ID: %d\n", s1.id);
   printf("Name: %s\n", s1.name);
   printf("Marks: %.2f\n", s1.marks);
   return 0;
}
```

3) Explain Union with syntax and example?

A **union** in C is a **user-defined data type** similar to a structure, but with one key difference:

All members share the same memory location.

Syntax of Union

```
union union_name {
   data_type member1;
   data_type member2;
   data_type member3;
};
```

Accessing Members

Use the dot operator (.)

d.i
d.f
d.ch

Example Program

```
#include <stdio.h>

union Data {
   int i;
   float f;
   char ch;
};

int main() {
   union Data d;

   d.i = 10;
   printf("Integer: %d\n", d.i);

   d.f = 5.5;
   printf("Float: %.2f\n", d.f);

   d.ch = 'A';
   printf("Character: %c\n", d.ch);

   return 0;
}
```

Output Explanation

- When d.i = 10 → memory stores integer
- When d.f = 5.5 → **same memory is overwritten**
- When d.ch = 'A' → previous values are lost

So, only the **last assigned value is valid**

4) Explain Enumerated data type with syntax and example

An **Enumerated Data Type (enum)** in C is a **user-defined data type** used to assign **names to a set of integer constants**, making programs more readable and meaningful.

Syntax

```
enum enum_name {
   value1, value2, value3, ...
};
```

Example 1 (Basic Enum)

```
#include <stdio.h>
// defining enum
enum Day {SUN, MON, TUE, WED, THU, FRI, SAT};
int main() {
   enum Day today;
   today = WED;
   printf("Value of today: %d\n", today);
   return 0;
}
```

Output

Value of today: 3

By default:
SUN = 0
MON = 1
TUE = 2
WED = 3 … and so on

UNIT-V

Object-Oriented Programming (OOP) is a programming approach that organizes code using objects and classes. It focuses on modeling real-world entities.

Main Features of OOP

1. Class

A class is a blueprint or template for creating objects.

Example:

```
class Student {
    int id;
    char name[20];
};
```

It defines properties (data) and behavior (functions).

2. Object

An object is an instance of a class.

Example:

```
Student s1;
```

s1 is an object of class Student.

3. Encapsulation

Encapsulation means wrapping data and functions together into a single unit (class).

It also helps in data hiding.

Example:

```
class Account {
private:
    int balance;
public:
    void setBalance(int b) {
        balance = b;
    }
};
```

4. Abstraction

Abstraction means showing only essential details and hiding internal implementation.

Focus on *what* an object does, not *how* it does it.

Example:

Using functions without knowing internal logic.

5. Inheritance

Inheritance allows one class to reuse properties and methods of another class.

Promotes code reusability.

Example:

```
class Animal {
public:
    void eat() {
        printf("Eating");
    }
};
```

```
class Dog : public Animal {
};
```

6. Polymorphism

Polymorphism means one name, many forms.

Types:

- Compile-time (Function Overloading)
- Run-time (Function Overriding)

Example:

```
int add(int a, int b);
float add(float a, float b);
```

2) Similarities and Differences between C & C++

Similarities between C & C++

1. **Syntax**
   - C++ is an extension of C, so both have very similar syntax.
2. **Procedural Nature**
   - Both support **procedural programming** (functions, step-by-step execution).
3. **Data Types**
   - Both use common data types like int, float, char, etc.
4. **Operators**
   - Arithmetic, relational, logical operators are almost the same.
5. **Control Structures**
   - if, else, for, while, switch are used in both.
6. **Functions**
   - Both support user-defined functions.
7. **Pointers**
   - Both languages use pointers for memory handling.

Differences between C & C++

Feature	C Language	C++ Language
Programming Type	Procedural	Object-Oriented + Procedural
Approach	Top-down	Bottom-up
Focus	Functions	Objects & Classes
Data Security	Less secure	More secure (Encapsulation)
OOP Features	Not supported	Supported (OOP concepts)
Classes & Objects	Not available	Available
Inheritance	Not supported	Supported
Polymorphism	Not supported	Supported
Function Overloading	Not available	Available
Exception Handling	Not available	Available (`try`, `catch`)
Input/Output	`printf()`, `scanf()`	`cin`, `cout`
Namespace	Not supported	Supported

3) Explain storage Class?

In C, **storage classes** define **how and where variables are stored**, their **lifetime (scope)**, and **visibility** in a program.

Types of Storage Classes in C

There are **four main storage classes**:

1. auto
2. register
3. static
4. extern

1. auto Storage Class

Definition

- Default storage class for local variables
- Stored in **memory (RAM)**
- Scope is **within the function/block**

Syntax:

```
auto int x;
```

Example:

```
#include <stdio.h>
int main() {
   auto int x = 10;
   printf("%d", x);
   return 0;
}
```

auto is optional (compiler assumes it by default)

2. register Storage Class

Definition

- Stores variable in **CPU register** (faster access)
- Used for **frequently accessed variables**

Syntax:

```
register int x;
```

Example:

```
#include <stdio.h>

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

Note: You cannot use & (address operator) with register variables

3. static Storage Class

Definition

- Retains value **between function calls**
- Initialized only once

Syntax:

```
static int x;
```

Example:

```
#include <stdio.h>

void counter() {
```

```
    static int count = 0;
    count++;
    printf("%d\n", count);
}

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

Output:

```
1
2
3
```

Value is preserved across function calls

4. extern Storage Class

Definition

- Used to declare a **global variable defined in another file or outside function**

Syntax:

```
extern int x;
```

Example:

```
#include <stdio.h>
int x = 50;  // global variable
int main() {
    extern int x;
    printf("%d", x);
    return 0;
}
```

Storage Class	Scope	Lifetime	Default Value	Storage Location
auto	Local	Function/block	Garbage	Memory (RAM)
register	Local	Function/block	Garbage	CPU Register
static	Local/Global	Entire program	0	Memory
extern	Global	Entire program	0	Memory

4) Explain Class-Object with syntax and example.

Class

A **class** is a **user-defined data type** that acts as a **blueprint** for creating objects.

It contains **data members (variables)** and **member functions (methods)**.

Syntax of Class

```
class ClassName {
private:
    // data members

public:
    // member functions
};
```

2. Object

An **object** is an **instance of a class**.

It represents a real-world entity and is used to access the class members.

Syntax of Object

```
ClassName object_name;
```

Example

```
#include <iostream>
using namespace std;

// class definition
class Student {
private:
    int id;
    float marks;

public:
    void setData(int i, float m) {
        id = i;
        marks = m;
    }

    void display() {
        cout << "ID: " << id << endl;
        cout << "Marks: " << marks << endl;
    }
};

int main() {
    Student s1;   // object creation

    s1.setData(101, 85.5);  // calling function
    s1.display();           // displaying data
    return 0;
}
```

Output

```
ID: 101
Marks: 85.5
```

5) Explain Inheritance and its types with types

Inheritance is an OOP concept where one class (**derived/child class**) acquires the **properties and behaviors** (data + functions) of another class (**base/parent class**).

It helps in **code reuse** and **extending existing programs**.

Basic Syntax

```
class BaseClass {
   // base class members
};

class DerivedClass : access_specifier BaseClass {
   // derived class members
};
```

Example Program

```
#include <iostream>
using namespace std;

class Animal {
public:
    void eat() {
        cout << "Animal is eating" << endl;
    }
};

class Dog : public Animal {
public:
    void bark() {
        cout << "Dog is barking" << endl;
    }
};

int main() {
    Dog d;
    d.eat();   // inherited function
    d.bark();  // own function
    return 0;
}
```

Output

```
Animal is eating
Dog is barking
```

Types of Inheritance

1. Single Inheritance

One derived class inherits from one base class.

```
class A { };
class B : public A { };
```

2. Multiple Inheritance

One derived class inherits from **more than one base class**.

```
class A { };
class B { };
class C : public A, public B { };
```

3. Multilevel Inheritance

A class is derived from another derived class.

```
class A { };
class B : public A { };
class C : public B { };
```

4. Hierarchical Inheritance

Multiple derived classes inherit from the same base class.

```
class A { };
class B : public A { };
class C : public A { };
```

5. Hybrid Inheritance

Combination of two or more types of inheritance.

```
class A { };
class B : public A { };
class C { };
class D : public B, public C { };
```

Short Answers Questions

1. What is a programming language?

A **programming language** is a formal language used to write instructions that a computer can understand and execute.

2. What are the types of programming languages?

- Machine language
- Assembly language
- High-level language

3. What is the history of C language?

C was developed by **Dennis Ritchie** in 1972 at Bell Laboratories for system programming.

4. What is the basic structure of a C program?

- Documentation
- Preprocessor directives
- Global declarations
- main() function
- Statements

5. What is an algorithm?

An **algorithm** is a step-by-step procedure to solve a problem.

6. What is a flowchart?

A **flowchart** is a graphical representation of an algorithm using symbols.

7. What are library functions?

Predefined functions provided by C to perform common tasks.
Example: printf(), scanf()

8. What is a preprocessor in C?

A **preprocessor** processes directives before compilation.
Example: #include, #define

9. What are keywords in C?

Keywords are **reserved words** with predefined meanings.
Example: int, if, return

10. What is an identifier?

An **identifier** is the name given to variables, functions, or arrays.

11. What are constants?

Constants are **fixed values** that do not change during program execution.
Example: 10, 'A', 3.14

12. What is a variable?

A **variable** is a named memory location used to store data.

13. What are the rules for defining variables?

- Must start with a letter or underscore
- Cannot use keywords
- No spaces allowed
- Case-sensitive

14. What is scope of a variable?

Scope defines **where a variable can be accessed** (local/global).

15. What is lifetime of a variable?

Lifetime is the **duration for which a variable exists in memory**.

16. What are data types?

Data types define the **type of data** a variable can store.
Example: int, float, char

17. What is type conversion?

Conversion of one data type into another.
Example: int to float

18. What is formatted input/output?

Using functions like printf() and scanf() with format specifiers.
Example: %d, %f

19. What is an operator in C?

An **operator** is a symbol used to perform operations on variables.

20. What are arithmetic operators?

Operators used for calculations.
Example: +, -, *, /, %

21. What are relational operators?

Used to compare values.
Example: ==, >, <

22. What are logical operators?

Used to combine conditions.
Example: &&, ||, !

23. What are assignment operators?

Used to assign values.
Example: =, +=, -=

24. What is a conditional operator?

A shorthand for if-else.
Syntax: condition ? expr1 : expr2;

25. What are bitwise operators?

Operate on binary values.
Example: &, |, ^, <<, >>

26. What are increment and decrement operators?

- ++ increases value by 1
- -- decreases value by 1

27. What are special operators?

Operators like sizeof, &, *, ,

28. How is a C program executed?

- Write code
- Compile (using compiler)
- Run (execute)

29. What is printf()?

Used to display output on screen.

30. What is scanf()?

Used to take input from the user.

Unit II

1. What are conditional statements?

Conditional statements are used to **make decisions** in a program based on conditions (true/false).

2. What is an `if` statement?

Executes a block of code **only if the condition is true**.
Syntax: `if(condition) { statements; }`

3. What is an `if-else` statement?

Executes one block if condition is true, otherwise another block.
Syntax: `if(cond) {} else {}`

4. What is a nested `if` statement?

An `if` statement inside another `if` statement.

5. What is a `switch` statement?

Used to select one block of code from multiple options based on an expression.

6. What is the use of `break` statement?

Terminates the loop or `switch` immediately.

7. What is `continue` statement?

Skips the current iteration and moves to the next iteration of the loop.

8. What is `goto` statement?

Transfers control to a labeled statement in the program. *(Not recommended in practice)*

9. What is the difference between `if` and `switch`?

- `if` → used for complex conditions
- `switch` → used for fixed values (constants)

10. Can we use `break` in `if` statement?

No, `break` is used only in loops and `switch`.

11. What is a loop?

A loop is used to **repeat a block of code** multiple times until a condition becomes false.

12. What is a `while` loop?

An entry-controlled loop that executes while condition is true.
Syntax: `while(condition) {}`

13. What is a `do-while` loop?

An exit-controlled loop that executes at least once.
Syntax: `do {} while(condition);`

14. What is a `for` loop?

A loop that combines initialization, condition, and increment.
Syntax: `for(init; cond; inc) {}`

15. What is a nested loop?

A loop inside another loop.

16. Difference between `while` and `do-while`?

- `while` → condition checked first
- `do-while` → condition checked after execution

17. Which loop executes at least once?

`do-while` loop.

18. Which loop is best when iterations are known?

`for` loop.

19. What is an infinite loop?

A loop that runs **forever** because the condition never becomes false.

20. Can we use `break` in loops?

Yes, it is used to exit the loop immediately.

21. Can we use `continue` in loops?

Yes, it skips the current iteration.

22. What is the use of nested loops?

Used when repetition is required within another repetition (e.g., matrix operations).

23. Give an example of a loop.

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

24. What is loop control variable?

A variable that controls the number of iterations in a loop.

25. What happens if condition is false in `while` loop initially?

The loop will **not execute even once**.

Unit III

1. What is a function in C?

A **function** is a block of code that performs a specific task and can be reused.

2. What is a function declaration?

It informs the compiler about the function name, return type, and parameters before its use.
Example: `int add(int, int);`

3. What is a function prototype?

A **function prototype** is a declaration of a function before `main()`.

4. What is a return statement?

It is used to **return a value from a function** to the calling function.
Example: `return x;`

5. What are the types of functions?

- Built-in (library) functions
- User-defined functions

6. What are formatted functions?

Functions that use **format specifiers** for input/output.
Example: `printf()`, `scanf()`

7. What are unformatted functions?

Functions that do not use format specifiers.
Example: `getchar()`, `putchar()`

8. What are mathematical functions?

Functions used for mathematical calculations from `<math.h>`.
Example: `sqrt()`, `pow()`

9. What are string functions?

Functions used to manipulate strings from `<string.h>`.
Example: `strlen()`, `strcpy()`

10. What are character functions?

Functions used to handle characters from `<ctype.h>`.
Example: `isalpha()`, `isdigit()`

11. What are date functions?

Functions used to handle date and time from `<time.h>`.
Example: `time()`, `ctime()`

12. What is a user-defined function?

A function created by the programmer to perform a specific task.

13. What is the need for user-defined functions?

- Code reusability
- Modularity
- Reduces complexity

14. What are the elements of a function?

- Function declaration
- Function definition
- Function call

15. What is a function call?

Calling a function to execute its code.
Example: `add(5, 3);`

16. What is call by value?

Passing a copy of variables to a function (original not affected).

17. What is call by reference?

Passing address of variables (original values are modified).

18. What is a recursive function?

A function that calls itself.
Example: factorial calculation

19. What is an array?

A collection of elements of the same data type stored in contiguous memory.

20. How do you define an array?

Syntax: `int arr[5];`

21. What is array initialization?

Assigning values to array elements.
Example: `int arr[3] = {1,2,3};`

22. What are characteristics of arrays?

- Same data type
- Fixed size
- Indexed access
- Continuous memory

23. What is a one-dimensional array?

An array with one index.
Example: `int arr[5];`

24. What is a two-dimensional array?

An array with rows and columns.
Example: `int arr[2][3];`

5. What is a multidimensional array?

An array with more than two dimensions.
Example: `int arr[2][2][2];`

26. What is a string in C?

A string is a collection of characters ending with `\0`.

27. How do you declare a string?

Syntax: `char str[10];`

28. How do you initialize a string?

Example: `char str[] = "Hello";`

29. How do you read a string?

Using `scanf()` or `gets()` *(deprecated)*
Example: `scanf("%s", str);`

30. How do you write a string?

Using `printf()`
Example: `printf("%s", str);`

31. What are string standard functions?

Functions used to manipulate strings.
Example: `strlen()`, `strcat()`, `strcmp()`

Unit IV

1. What is a pointer?

Answer:
A pointer is a variable that stores the **address of another variable**.

2. How do you declare a pointer?

Answer:

data_type *pointer_name;

Example: int *p;

3. What is the use of & operator?

Answer:
It is used to **get the address** of a variable.

4. What is the use of * operator?

Answer:
It is used to **access the value** stored at a memory address (dereferencing).

5. What are the features of pointers?

Answer:

- Stores address of variables
- Supports dynamic memory allocation
- Used in arrays and functions
- Enables call by reference

6. What is pointer arithmetic?

Answer:
Operations performed on pointers like **increment, decrement, addition, subtraction**.

7. Give an example of pointer arithmetic.

Answer:

p = p + 1;

Moves pointer to next memory location.

8. What is a structure?

Answer:
A structure is a **user-defined data type** used to group different types of data.

9. Write syntax of structure.

Answer:

```
struct name {
   data_type member;
};
```

10. How to access structure members?

Answer:
Using **dot (.) operator**. Example: s1.id

11. What are the features of structures?

Answer:

- Groups different data types
- Improves data organization
- Supports arrays and nesting

12. What is nested structure?

Answer:
A structure inside another structure.

13. What is an array of structures?

Answer:
Collection of structure variables stored in an array.

Example:

struct Student s[10];

14. What is enum?

Answer:
An enum is a user-defined data type that assigns **names to integer constants**.

15. What is default value of enum?

Answer:
Starts from **0** by default.

16. What is a union?

Answer:
A union is a data type where **all members share the same memory location**.

17. What is the size of union?

Answer:
Size of the **largest member**.

18. What are advantages of unions?

Answer:

- Saves memory
- Efficient memory usage
- Useful when only one value is needed at a time

19. Difference between structure and union?

Answer:

Structure	**Union**
Separate memory	Shared memory
All members usable	One member at a time
Larger size	Smaller size

20. What is call by reference using pointers?

Answer:
Passing address of variables to function so changes affect original values.

21. Can we store multiple values in union at same time?

Answer:
No, only one value can be stored at a time.

Unit V

1. What is Object-Oriented Programming (OOP)?

Answer:
OOP is a programming approach that uses **objects and classes** to design programs and model real-world entities.

2. What are the main features of OOP?

Answer:

- Encapsulation
- Abstraction
- Inheritance
- Polymorphism

3. What is the basic structure of a C++ program?

Answer:

- Header files
- Namespace
- Class definition
- Main function
- Program statements

4. Write a simple C++ program.

Answer:

```
#include <iostream>
using namespace std;

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

5. What are storage classes?

Answer:
Storage classes define the **scope, lifetime, and visibility** of variables.

6. Name storage classes in C++.

Answer:

- auto
- register
- static
- extern

7. Give one similarity between C and C++.

Answer:
Both have similar syntax and support functions.

8. Give one difference between C and C++.

Answer:
C is procedural, while C++ supports object-oriented programming.

9. What are data members?

Answer:
Variables declared inside a class.

10. What are member functions?

Answer:
Functions defined inside a class to operate on data members.

11. What is a class?

Answer:
A class is a blueprint for creating objects.

12. What is an object?

Answer:
An object is an instance of a class.

13. What is inheritance?

Answer:
It is the process by which one class acquires properties of another class.

14. Name any two types of inheritance.

Answer:

- Single inheritance
- Multiple inheritance

15. What is polymorphism?

Answer:
Polymorphism means **one function or operator behaving differently in different situations**.

16. Name types of polymorphism.

Answer:

- Compile-time
- Run-time

17. What is encapsulation?

Answer:
Wrapping data and functions together into a single unit (class).

18. What is abstraction?

Answer:
Showing only essential features and hiding implementation details.

19. What is using namespace std?

Answer:
It allows using standard library names without prefixing std::.

20. What is the use of main() function?

Answer:
It is the entry point of a C++ program.