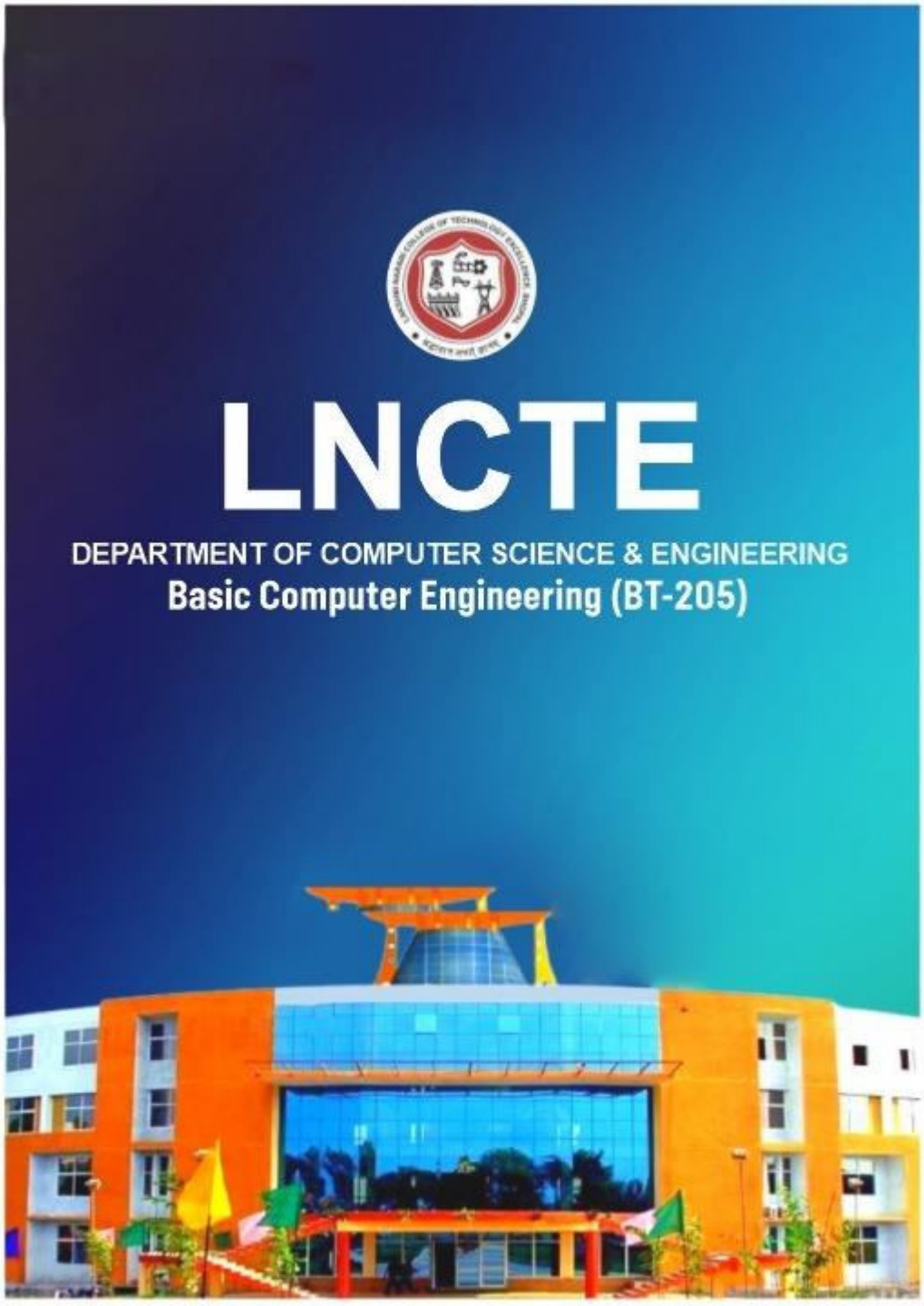
LNCTE
DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING
Basic Computer Engineering (BT-205)

Lakshmi Narain College of Technology Excellence

Department of Computer Science and Engineering

Lab Manual

Subject Name: Basic Computer Engineering

Course Code: BT205

Course: B.Tech

Session: 2023-24

Prepared By

TABLE OF CONTENT

Vision and Mission of the Institute

Vision of the institute

To become a pioneer institute in technical education and innovations to build competent technocrats and leaders for the nation.

Mission of the institute

M1. To enhance the academic environment with innovative teaching learning processes and modern tools.

M2. To Practice and nurture high standards of human values, transparency and accountability.

M3. To collaborate with other academic and research institutes as well as industries in order to strengthen education and research.

M4. To uphold skill development for employability and entrepreneurship for interdisciplinary research and innovations.

Vision and Mission of the Department

Vision of the Department

To be a centre of excellence for providing quality technical education to develop future leaders with the aspects of research & computing, Software product development and entrepreneurship.

Mission of the Department

Mission No.	Mission Statements
M1	To offer academic program with state of art curriculum having flexibility for accommodating the latest developments in the areas of computer science engineering.
M2	To conduct research and development activities in contemporary and emerging areas of computer science & engineering.
M3	To inculcate moral values & entrepreneurial skills to produce professionals capable of providing socially relevant and sustainable solutions.

COURSE OUTCOMES: BT205 – Basic Computer Engineering

CO205.1	Illustrate basic commands and operations in operating system
CO205.2	Implement basic programming concepts with C++.
CO205.3	Understand various object oriented features like polymorphism, inheritance, object, classes.
CO205.4	Study and analyze OSI models as well as computer security issues.
CO205.5	Learn and illustrate DBMS fundamental concepts.

Course Articulation Matrix

PO CO	**PO1**	**PO2**	**PO3**	**PO4**	**PO5**	**PO6**	**PO7**	**PO8**	**PO9**	**PO10**	**PO11**	**PO12**
CO205.1	3	3	3	3	-	1	-	1	2	1	-	3
CO205.2	3	3	3	3	-	1	-	-	2	1	-	3
CO205.3	3	2	3	3	-	1	-	-	1	1	1	3
CO205.4	3	3	3	3	-	1	-	1	2	1	-	3
CO205.5	3	3	3	3	1	1	-	-	2	1	1	3
	3	**2.8**	**3**	**3**	**1**	**1**	**-**	**1**	**1.8**	**1**	**1**	**3**

Program Outcomes as defined by NBA (PO)

Engineering Graduates will be able to:

1. Engineering knowledge: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.

2. Problem analysis: Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.

3. Design/development of solutions: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.

4. Conduct investigations of complex problems: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.

5. Modern tool usage: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.

6. The engineer and society: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.

7. Environment and sustainability: Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.

8. Ethics: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.

9. Individual and team work: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.

10. Communication: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.

11. Project management and finance: Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.

12. Life-long learning: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

Department of Computer Science and Engineering

Program Specific Outcomes (PSO)

A graduate of Computer Science and Engineering Program will develop

PSO1: An ability to apply technical knowledge of computer science and engineering fundamentals to become employable in industry.

PSO2: An ability to develop programming skills using modern software tools and techniques.

PSO3: An ability to develop real time projects for problem solving of domains such as Machine Learning, Cyber security, block chain and big data.

PSO4: An ability to grab research, higher studies and entrepreneurship opportunities towards society with moral values and ethics.

Department of Computer Science and Engineering

Program Educational Objectives (PEO):

PEO 1: Evolve as globally competent computer professionals, researchers and entrepreneurs possessing collaborative and leadership skills, for developing innovative solutions in multidisciplinary domains.

PEO 2: Excel as socially committed computer engineers having mutual respect, effective communication skills, high ethical values and empathy for the needs of society.

PEO 3: Involve in lifelong learning to foster the sustainable development in the emerging areas of technology

Department of Computer Science and Engineering

Basic Computer Engineering (BT205)

List of Program to be performed: -

Lab No.	List of Program	CO Covered
1	Study and Practice Internal and External DOS commands.	(CO1)
2	Study and Practice of MS windows-Folder related operations, My computer, Window Explore, Control Panel	(CO1)
3	Creating and operating of spreadsheet using MS Excel	(CO1)
4	Creation and manipulation of database table using SQL in MS-Access	(CO1)
5	WAP to illustrate various arithmetic functions using functions like add(),sub(),multi(), div() etc.	(CO1)
6	WAP to take 10 numbers in any array and print sum of that numbers.	(CO2)
7	WAP to add two numbers using function	(CO2)
8	WAP using class to illustrate concept of Constructor and Destructor. Also try to use scope resolution program	(CO3)
9	WAP to implement operator overloading like “+” operator.	(CO3)
10	WAP to implement runtime polymorphism.	(CO3)
11	Study and analyze OSI models as well as computer security issues	(CO4)
12	Creating and manipulation of Database table using SQL in MS Access.	(CO5)

Basic Computer Engineering BT205

Computer engineering is a multidisciplinary field that integrates principles from electrical engineering and computer science to design, develop, and optimize computer systems. At its core, it involves understanding digital logic circuits, encompassing basic logic gates and more complex structures. Computer architecture plays a pivotal role, exploring the organization of key components such as the Central Processing Unit (CPU), memory, and input/output devices. Proficiency in programming languages, including high-level ones like C++, Java, or Python, is essential. Operating systems knowledge is crucial, covering processes, memory management, and file systems. Additionally, computer engineers delve into computer networks, studying protocols, routing, and layers. Data structures and algorithms form fundamental building blocks for efficient software development. Knowledge of embedded systems, VLSI for integrated circuits, and computer security principles further enriches the skill set. As technology evolves, areas like digital signal processing and computer graphics become increasingly relevant. The dynamic nature of computer engineering requires continuous learning and adaptation to stay abreast of advancements in this rapidly evolving field.

OOPs Concepts:

- Class
- Objects
- Data Abstraction
- Encapsulation
- Inheritance
- Polymorphism
- Dynamic Binding
- Message Passing

EXPERIMENT-1

Aim: Study and practice of Internal & External DOS commands.

The study and practice of DOS (Disk Operating System) commands involve understanding and utilizing commands to interact with the computer's operating system. DOS commands can be categorized into internal and external commands.

Internal Commands:

Internal commands are built into the command interpreter (usually COMMAND.COM or CMD.EXE in Windows). They don't require separate executable files and are directly interpreted by the command shell. Some common internal commands include:

1. DIR: Displays a list of files and subdirectories in a directory.
2. CD (or CHDIR): Changes the current directory.
3. CLS: Clears the screen.
4. COPY: Copies one or more files to another location.
5. DEL (or ERASE): Deletes one or more files.
6. REN (or RENAME): Renames a file or directory.
7. TYPE: Displays the contents of a text file.
8. MKDIR (or MD): Creates a new directory.
9. RMDIR (or RD): Removes a directory.
10. DATE: Displays or sets the date.

External Commands:

External commands are separate executable files stored in directories listed in the PATH environment variable. These commands extend the functionality of the operating system. Examples of external commands include:

1. FORMAT: Prepares a disk for use by an operating system.
2. CHKDSK: Checks a disk for errors and fixes them.
3. XCOPY: Copies files and directory trees with more options than the COPY command.
4. DISKPART: Manages disk partitions on a hard drive.
5. PING: Tests network connectivity.
6. IPCONFIG: Displays network configuration information.
7. TASKKILL: Terminates processes or applications by name or process ID.
8. TREE: Displays the structure of a directory or path graphically.

9. EDIT: Opens a simple text editor for creating or editing text files.

To practice DOS commands, you can open the Command Prompt on a Windows system and start experimenting with these commands. Use the help command followed by a specific command to get more information about its usage and options. Additionally, online resources and tutorials can provide hands-on exercises and scenarios to enhance your understanding of DOS commands and their practical applications

Viva Voce questions-

- How can DOS commands be used in batch files for automation?
- Demonstrate how to use the DIR command to list files in a specific directory.
- Explain the functionality of the FORMAT command in DOS.
- How does the XCOPY command differ from the internal COPY command?

EXPERIMENT-2

Aim: Study and Practice of MS windows – Folder related operations, My-Computer, window explorer, Control Panel

My Computer, Windows Explorer, and Control Panel is essential for effective navigation and management in a Windows environment.

1. Folder Related Operations

- **Creating a New Folder**

Right-click in the desired location.

Choose "New" and then "Folder".

Give the folder a name.

- **Renaming a Folder:**

Right-click on the folder

Select "Rename" and type the new name.

- **Copying and Moving Folders**

Use "Ctrl + C" to copy and "Ctrl + V" to paste.

Alternatively, use the right-click context menu for "Copy" and "Paste."

For moving, you can either drag and drop or use "Cut" and "Paste."

- **Deleting a Folder**

Right-click on the folder.

Select "Delete" or use the "Delete" key.

2. My Computer:

- **Accessing My Computer:**

On the desktop, double-click the "My Computer" or "Computer" icon.

Alternatively, press "Windows key + E" to open Windows Explorer to the "Computer" view.

- **Viewing Drives and Storage:**

Displays all available drives (hard drives, SSDs, external drives).

Shows storage space and provides quick access to drive contents.

- **Mapping Network Drives:**

Right-click on a network drive.

Select "Map Network Drive" to assign a drive letter.

3. Windows Explorer:

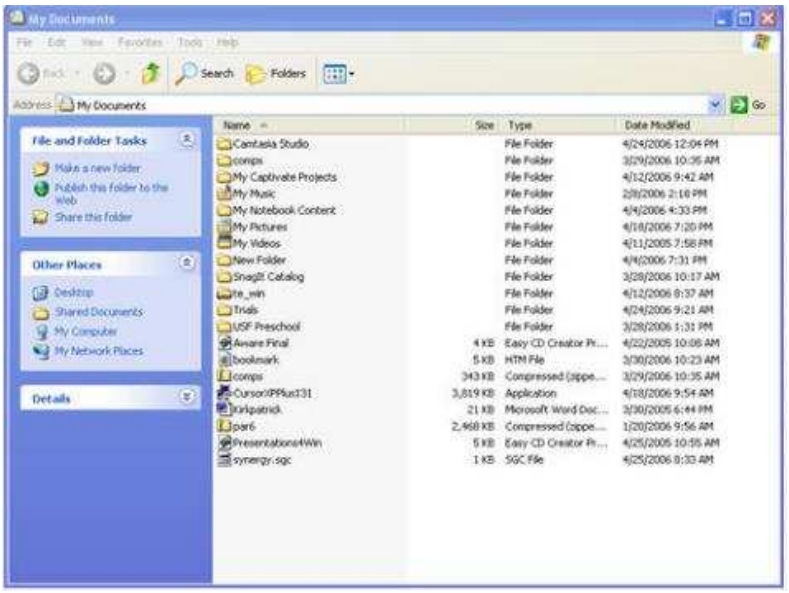

Figure 1: Window Explore

- **Navigating Through Folders:**

Use the left pane for navigation.

Double-click folders to enter them.

- **Searching for Files:**

  Use the search bar in the top-right corner.

  Enter file names or keywords to find files quickly

- **Sorting and Arranging Files:**

  Click on column headers to sort files (name, date, size).

  Use the "View" tab to arrange files by details, tiles, icons, etc.

- **Accessing Quick Access:**

  Quick Access shows frequently accessed folders and recent files.

4. Control Panel:

Figure 2: Control Panel

- **Accessing Control Panel:**

  Open the Start menu and select "Control Panel."

  In Windows 10 and later, you can also use the Settings app.

- **Changing System Settings:**

  Adjust system settings, such as display resolution, power options, etc

  Access "System and Security" for firewall, antivirus, and maintenance settings.

- **Managing User Accounts:**

  Add or remove user accounts.

  Modify account settings, passwords, and permissions.

- **Installing/Uninstalling Programs:**

  Access "Programs" or "Programs and Features" to install or uninstall software.

Viva Voce Questions-

- Explain the steps to create a new folder on the desktop.
- Differentiate between Windows Explorer and My Computer.
- What is the Control Panel, and why is it important in Windows?
- How do you troubleshoot common issues using the Control Panel?

EXPERIMENT-3

Aim: Creation and operating of spreadsheet using MS-Excel

Creating and operating a spreadsheet in Microsoft Excel involves several steps.

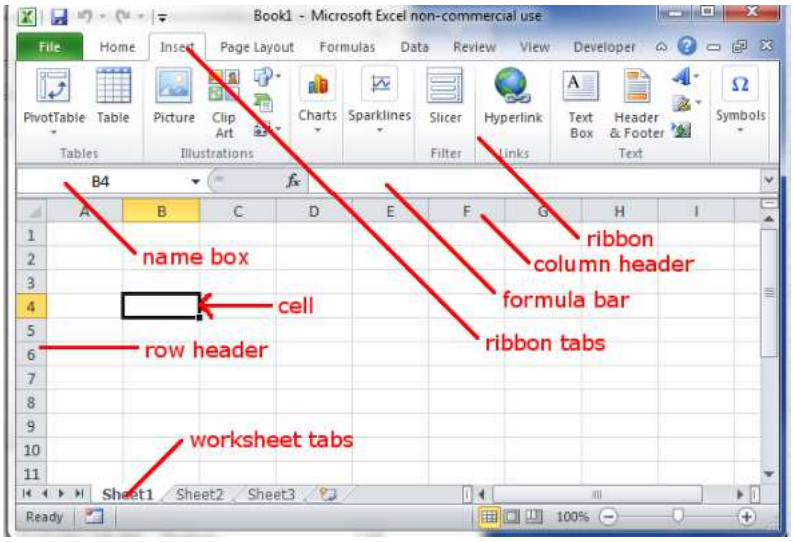

Figure 5: Terms used in MS-Excel

Open Microsoft Excel

Launch Microsoft Excel on your computer

Create a New Workbook

You'll typically start with a new workbook, which is like a container for your spreadsheet.

File > New > Blank Workbook

Understanding the Excel Interface

Familiarize yourself with the Excel interface, including the Ribbon, Cells, Columns, Rows, and Sheets.

Enter Data

Click on a cell and start typing to enter data

You can use the Tab key to move to the next cell or Enter to move down

Formatting Cells

You can format cells by changing font size, color, alignment, and more.

Right-click on a cell or use the options in the Ribbon (Home tab) for formatting.

Managing Rows and Columns

Inserting Rows/Columns: Right-click on a row/column header and choose "Insert."

Deleting Rows/Columns: Right-click on a row/column header and choose "Delete."

Formulas and Functions

Use formulas to perform calculations

Type a formula in a cell, e.g., =A1+B1, and press Enter

Explore functions (e.g., SUM, AVERAGE) from the "Formulas" tab.

Autofill

Drag the fill handle (a small square at the bottom right of the selected cell) to copy data or create a series

Cell References

Understand relative (A1), absolute (A1), and mixed (A$1, $A1) cell references in formulas.

Charts and Graphs

Highlight data and go to the "Insert" tab to create charts or graphs.

Sorting and Filtering

Use the "Sort" and "Filter" options (Home tab) to organize and analyze data.

Data Validation

Restrict the type of data that can be entered in a cell using data validation.

Protecting Sheets and Workbooks

Password-protect sheets or the entire workbook to prevent unauthorized changes.

Saving Your Workbook

Save your work regularly using Ctrl + S or File > Save.

Printing

Use the "Print" option (File > Print) to print your spreadsheet.

Excel Shortcuts

Learn some useful keyboard shortcuts to speed up your work.

Viva Voce Questions-

- Differentiate between a formula and a function in Excel.
- What is the purpose of the SUM function? Can you provide an example?
- How do you create a bar chart in Excel?
- How can you password-protect a worksheet in Excel?
- Explain the concept of relative and absolute cell references.

EXPERIMENT-4

Aim: Creation and manipulation of database table using SQL in MS-Access.

Creating and manipulating database tables using SQL in Microsoft Access involves various SQL commands.

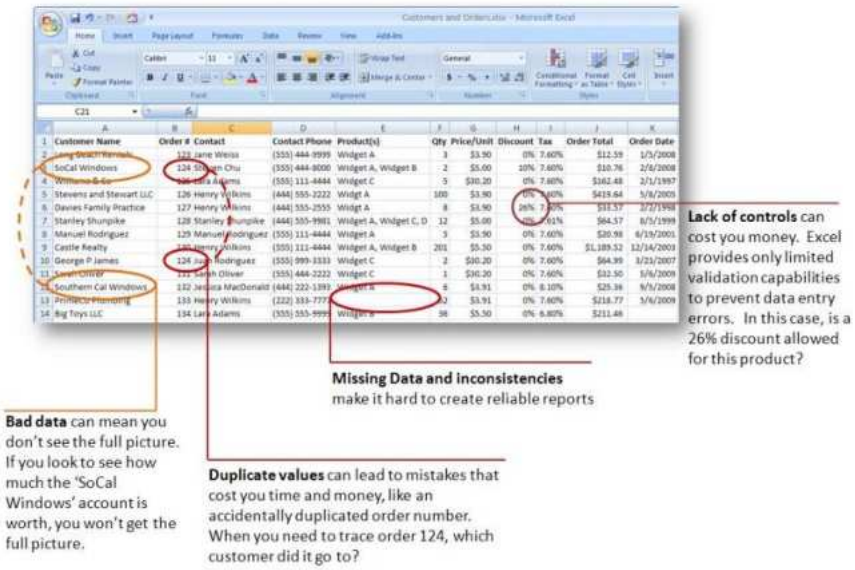

Figure 6: Terms used in MS-Access

Creating a Table

Open Microsoft Access

Launch Microsoft Access on your computer.

Create a New Database

Click on "File" > "New" > "Blank Database" to create a new database

Create a New Table

Go to the "Table Design" view to manually create a table.

Define the fields (columns) and their data types

```
mysql> USE our_database;
Database changed
mysql> CREATE TABLE Employee_details(
    -> EmployeeID int Primary Key,
    -> Name varchar(255) NOT NULL,
    -> Address varchar(255),
    -> Salary decimal(18, 2)
    -> );
Query OK, 0 rows affected (1.77 sec)
```

Figure 7: Create table in SQL

Save the Table

Save the table by giving it a name and clicking on the "Save" button

SQL Commands for Table Creation

```
-- Example: Creating a table named "Employees"
CREATE TABLE Employees (
    EmployeeID INT PRIMARY KEY,
    FirstName VARCHAR(50),
    LastName VARCHAR(50),
    BirthDate DATE,
    Department VARCHAR(50)
);
```

Manipulating Data

Inserting Data

Use the INSERT INTO statement to add data to the table

```
-- Example: Inserting data into the "Employees" table
INSERT INTO Employees (EmployeeID, FirstName, LastName, BirthDate, Department)
VALUES (1, 'John', 'Doe', '1990-01-15', 'IT');
```

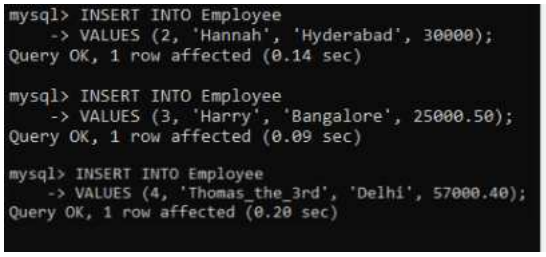

Figure 8: Inserting table in SQL

Selecting Data

Use the SELECT statement to retrieve data from the table

```
-- Example: Selecting all data from the "Employees" table
SELECT * FROM Employees;
```

Updating Data

Use the UPDATE statement to modify existing data in the table.

```
-- Example: Updating the department for employee with ID 1
UPDATE Employees
SET Department = 'Finance'
WHERE EmployeeID = 1;
```

Deleting Data

Use the DELETE FROM statement to remove data from the table

```
-- Example: Deleting an employee with ID 1 from the "Employees" table
DELETE FROM Employees
WHERE EmployeeID = 1;
```

SQL Commands for Data Manipulation:

```
-- Inserting data
INSERT INTO Employees (EmployeeID, FirstName, LastName, BirthDate, Department)
VALUES (2, 'Jane', 'Smith', '1985-05-20', 'HR');
-- Selecting data
SELECT * FROM Employees;
-- Updating data
UPDATE Employees
SET BirthDate = '1988-11-30'
WHERE EmployeeID = 2;
-- Deleting data
DELETE FROM Employees
WHERE EmployeeID = 2;
```

Relationships and Constraints

Defining Relationships

Use the Relationships window to establish relationships between tables.

Applying Constraints

Apply constraints like PRIMARY KEY, FOREIGN KEY, NOT NULL, etc., during table creation

```
-- Example: Adding a FOREIGN KEY constraint
CREATE TABLE Departments (
    DepartmentID INT PRIMARY KEY,
    DepartmentName VARCHAR(50)
);

CREATE TABLE Employees (
    EmployeeID INT PRIMARY KEY,
    FirstName VARCHAR(50),
    LastName VARCHAR(50),
```

```
    BirthDate DATE,
    DepartmentID INT,
    FOREIGN KEY (DepartmentID) REFERENCES Departments(DepartmentID)
);
```

Viva Voce Questions-

- Write an SQL query to insert a new customer record into a table named "Customers"
- Explain the purpose of the WHERE clause in SQL queries for data manipulation
- Explain the process of updating data in a table using SQL. Provide an example.
- Explain the purpose of the WHERE clause in SQL queries for data manipulation

EXPERIMENT-5

Aim:- WAP to illustrate Arithmetic expressions

Arithmetic expressions involve mathematical operations on numeric values. These expressions follow the rules of arithmetic and can include addition, subtraction, multiplication, division, modulus (remainder), and exponentiation

Addition (+)

Adds two values together

Example: a + b

Subtraction (-):

Subtracts the second value from the first

Example: a – b

Multiplication (*):

Multiplies two values

Example: a * b

Division (/):

Divides the first value by the second.

Example: a / b

Integer Division (//):

Performs division and truncates the result to the nearest integer.

Example: a // b

Modulus (%):

Returns the remainder after division.

Example: a % b

Exponentiation (^ or):

Raises the first value to the power of the second

Example: a ^ b or a ** b

Program

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

int main() {
   // Arithmetic Expressions Illustration
```

```
// Addition
int num1 = 10;
int num2 = 5;
int sum_result = num1 + num2;
cout << "Addition: " << num1 << " + " << num2 << " = " << sum_result << endl;

// Subtraction
int sub_result = num1 - num2;
cout << "Subtraction: " << num1 << " - " << num2 << " = " << sub_result << endl;

// Multiplication
int mul_result = num1 * num2;
cout << "Multiplication: " << num1 << " * " << num2 << " = " << mul_result << endl;

// Division
double div_result = static_cast<double>(num1) / num2;  // Ensuring a floating-point result
cout << "Division: " << num1 << " / " << num2 << " = " << div_result << endl;

// Integer Division
int int_div_result = num1 / num2;
cout << "Integer Division: " << num1 << " / " << num2 << " = " << int_div_result << endl;

// Modulus (Remainder)
int mod_result = num1 % num2;
cout << "Modulus: " << num1 << " % " << num2 << " = " << mod_result << endl;

// Exponentiation
int exp_result = 1;
for (int i = 0; i < num2; ++i) {
    exp_result *= num1;
}
cout << "Exponentiation: " << num1 << " ^ " << num2 << " = " << exp_result << endl;
```

```
    return 0;
}
```

Ouput:

Addition: 10 + 5 = 15

Subtraction: 10 - 5 = 5

Multiplication: 10 * 5 = 50

Division: 10 / 5 = 2

Integer Division: 10 / 5 = 2

Modulus: 10 % 5 = 0

Exponentiation: 10 ^ 5 = 100000

Viva Voce Questions-

- Explain the order of operations in arithmetic expressions.
- Explain the purpose of the subtraction operator (-) in arithmetic.
- In what scenarios would you use parentheses in an arithmetic expression?
- Describe the function of the division operator (/) in arithmetic.

EXPERIMENT-6

Aim: WAP to illustrate Arrays

An array is a data structure consisting of a collection of elements (values or variables), of same memory size, each identified by at least one array index or key. An array is stored such that the position of each element can be computed from its index tuple by a mathematical formula. Arrays are fundamental data structures used in programming to store a collection of elements of the same data type. These elements are stored in contiguous memory locations, and each element is accessed using an index or a key. The concept of arrays is widely used across various programming languages.

Steps:-

- Declares an array named numbers with five integer elements.
- Prints the elements of the array using a loop.
- Calculates the sum of all elements in the array.
- Finds the maximum element in the array.

Program

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

int main() {
   // Illustrating Arrays in C++

   // Declaration and Initialization
   int numbers[5] = {10, 20, 30, 40, 50};

   // Accessing and Printing Array Elements
   cout << "Array Elements: ";
   for (int i = 0; i < 5; ++i) {
      cout << numbers[i] << " ";
   }
   cout << endl;

   // Summing Array Elements
   int sum = 0;
```

```
    for (int i = 0; i < 5; ++i) {
        sum += numbers[i];
    }
    cout << "Sum of Array Elements: " << sum << endl;

    // Finding the Maximum Element
    int maxElement = numbers[0];
    for (int i = 1; i < 5; ++i) {
        if (numbers[i] > maxElement) {
            maxElement = numbers[i];
        }
    }
    cout << "Maximum Element: " << maxElement << endl;

    return 0;
}
```

Output:

Array Elements: 10 20 30 40 50

Sum of Array Elements: 150

Maximum Element: 50

Viva Voce Questions-

- Explain the concept of indexing in arrays.
- How do you declare an array in C++? Provide an example.
- How do you declare and initialize a two-dimensional array in C++?
- How can you create an array of user-defined data types in C++?

EXPERIMENT-7

Aim:- WAP to illustrate functions.

Functions:- A function is a block of code which only runs when it is called.

Functions are used to perform certain actions, and they are important for reusing code: Define the code once, and use it many times.

Create a Function

C++ provides some pre-defined functions, such as main(), which is used to execute code.

Syntax

```
void myFunction() {
  // code to be executed
}
```

Program

```
#include <iostream>

// Function to calculate the square of a number
double calculateSquare(double number) {
    double square = number * number;
    return square;
}

// Function to print the square of a number
void printSquare(double number) {
    double square = calculateSquare(number);
    std::cout << "The square of " << number << " is: " << square << std::endl;
}

// Main program
int main() {
    // Get user input for a number
    std::cout << "Enter a number: ";
    double userInput;
    std::cin >> userInput;
```

```
    // Call the printSquare function to calculate and print the square
    printSquare(userInput);

    return 0;
}
```

Output:

Enter a number: 12

The square of 12 is: 144

In this example

The calculateSquare function takes a number as an argument, calculates its square, and returns the result.

The printSquare function takes a number, calls calculateSquare to get the square, and then prints the result using std::cout.

The main program takes user input for a number and calls the printSquare function to calculate and display the square.

Viva Voce Questions-

- Explain the components of a function definition.
- What is a parameter in a function?
- Explain the concept of a default argument in C++.
- Explain the difference between pass by value and pass by reference.

EXPERIMENT-8

Aim: WAP to illustrate constructor & Destructor

Constructor: - Constructor in C++ is a special method that is invoked automatically at the time of object creation. It is used to initialize the data members of new objects generally. The constructor in C++ has the same name as the class or structure. It constructs the values i.e. provides data for the object which is why it is known as constructor.

Constructor is a member function of a class, whose name is same as the class name.

• Constructor is a special type of member function that is used to initialize the data members for an object of a class automatically, when an object of the same class is created.

• Constructor is invoked at the time of object creation. It constructs the values i.e. provides data for the object that is why it is known as constructor.

• Constructor do not return value, hence they do not have a return type.

```
The prototype of the constructor looks like
    <class-name> (list-of-parameters);
```

Destructor :- Destructor is an instance member function that is invoked automatically whenever an object is going to be destroyed. Meaning, a destructor is the last function that is going to be called before an object is destroyed.

- A destructor is also a special member function like a constructor. Destructor destroys the class objects created by the constructor.
- Destructor has the same name as their class name preceded by a tilde (~) symbol.
- It is not possible to define more than one destructor.
- The destructor is only one way to destroy the object created by the constructor. Hence destructor can-not be overloaded.
- Destructor neither requires any argument nor returns any value.
- It is automatically called when an object goes out of scope.
- Destructor release memory space occupied by the objects created by the constructor.
- In destructor, objects are destroyed in the reverse of an object creation.

The syntax for defining the destructor within the class:

```
~ <class-name>() {
   // some instructions
}
```

The syntax for defining the destructor outside the class:

```
<class-name> :: ~<class-name>() {
    // some instructions
}
```

Program

```
#include <iostream>

class MyClass {
public:
    // Constructor
    MyClass() {
        std::cout << "Constructor called" << std::endl;
    }

    // Destructor
    ~MyClass() {
        std::cout << "Destructor called" << std::endl;
    }

    // Other member functions or variables can be added here
};

int main() {
    // Creating an object of MyClass
    MyClass myObject;

    // You can perform other operations with myObject here

    // Destructor will be called when myObject goes out of scope

    return 0;
}
```

In this example

The MyClass class has a constructor, denoted by the same name as the class, and a destructor, denoted by a tilde (~) followed by the class name.

In the main function, an object myObject of type MyClass is created. When the object is created, the constructor is called, and when it goes out of scope (at the end of the main function), the destructor is called.

Output

Constructor called

Destructor called

Viva Voce Questions-

- How is a constructor different from other member functions?
- What is a destructor?
- How is the order of constructor and destructor calls determined?
- Why might you need a copy constructor?

EXPERIMENT-9

Aim: WAP to illustrate Operator overloading

Operator overloading :- C++ has the ability to provide the operators with a special meaning for a data type, this ability is known as operator overloading. Operator overloading is a compile-time polymorphism. For example, we can overload an operator '+' in a class like String so that we can concatenate two strings by just using +. Other example classes where arithmetic operators may be overloaded are Complex Numbers, Fractional Numbers, Big integers, etc

Program

```
#include <iostream>
class Complex {
private:
    double real;
    double imag;

public:
    // Constructor
    Complex(double r = 0.0, double i = 0.0) : real(r), imag(i) {}

    // Overloading the '+' operator to add two complex numbers
    Complex operator+(const Complex& other) const {
        return Complex(real + other.real, imag + other.imag);
    }

    // Overloading the '-' operator to subtract two complex numbers
    Complex operator-(const Complex& other) const {
        return Complex(real - other.real, imag - other.imag);
    }

    // Overloading the '*' operator to multiply two complex numbers
    Complex operator*(const Complex& other) const {
        return Complex(
            real * other.real - imag * other.imag,
            real * other.imag + imag * other.real
```

```
        );
    }

    // Overloading the '==' operator to compare two complex numbers
    bool operator==(const Complex& other) const {
        return (real == other.real) && (imag == other.imag);
    }

    // Display the complex number
    void display() const {
        std::cout << real << " + " << imag << "i" << std::endl;
    }
};

int main() {
    // Creating two complex numbers
    Complex complex1(2.0, 3.0);
    Complex complex2(1.0, 4.0);

    // Adding two complex numbers using operator overloading
    Complex sum = complex1 + complex2;

    // Subtracting two complex numbers using operator overloading
    Complex difference = complex1 - complex2;

    // Multiplying two complex numbers using operator overloading
    Complex product = complex1 * complex2;

    // Comparing two complex numbers using operator overloading
    bool isEqual = (complex1 == complex2);

    // Displaying the results
    std::cout << "Sum: ";
```

```
    sum.display();

    std::cout << "Difference: ";
    difference.display();

    std::cout << "Product: ";
    product.display();

    std::cout << "Are complex1 and complex2 equal? " << (isEqual ? "Yes" : "No") << std::endl;

    return 0;
}
```

Output:

Sum: 3 + 7i

Difference: 1 + -1i

Product: -10 + 11i

Are complex1 and complex2 equal? No

In this example:

The Complex class represents complex numbers with real and imaginary parts.

Operator overloading is used for +, -, *, and == operators

The display function is used to print the complex number.

Viva Voce Questions-

- What is operator overloading in C++
- Can you provide an example of overloading the equality (==) operator
- What is the difference between a binary and a unary operator in the context of operator overloading?
- Can operators be overloaded as static member functions?

EXPERIMENT-10

Aim:- WAP to implement runtime polymorphism

Runtime polymorphism, also known as dynamic polymorphism or late binding, is a concept in object-oriented programming (OOP) where the method or function that gets executed is determined at runtime rather than at compile time. This is achieved through the use of virtual functions or abstract classes.

Inheritance: First, there needs to be a relationship between classes, typically through inheritance. In OOP, you have a base class (or interface in some languages) and one or more derived classes that inherit from the base class.

Program

```
class Animal {
public:
    virtual void makeSound() {
        cout << "Generic animal sound" << endl;
    }
};

class Dog : public Animal {
public:
    void makeSound() override {
        cout << "Woof!" << endl;
    }
};
```

Virtual Function: The base class should declare at least one virtual function. In the example above, makeSound() is declared as a virtual function in the Animal class.

Override in Derived Classes: In the derived class (Dog in this case), the virtual function is overridden to provide a specific implementation for that class.

Pointer or Reference to Base Class: You can use a pointer or reference of the base class type to refer to objects of the derived class. This allows for dynamic method binding.

```
Animal* myAnimal = new Dog();
myAnimal->makeSound();  // Calls Dog's makeSound() at runtime
```

The use of virtual functions enables the compiler to defer the method resolution until runtime, making it possible to achieve polymorphic behavior. This is particularly useful in scenarios where

you want to write code that can work with objects of multiple derived types through a common interface.

Viva Voce Questions-

- What is polymorphism?
- What are the advantages of polymorphism?
- Explain types of polymorphism?
- How does polymorphism differ from function overloading in C++?

EXPERIMENT-11

Aim: - Study and analyse OSI models as well as computer security issues

The OSI (Open Systems Interconnection) model is a conceptual framework that standardizes the functions of a telecommunication or computing system into seven abstraction layers. These layers, from the physical transmission of bits to the user interface, include the Physical, Data Link, Network, Transport, Session, Presentation, and Application layers. The model facilitates interoperability between different systems by providing a clear and modular structure for understanding and designing network protocols and communication processes

In the context of computer security, the OSI model plays a crucial role in identifying potential vulnerabilities and implementing security measures across various layers. Each layer presents distinct security challenges. The Physical layer may involve securing physical access to network components, while the Data Link and Network layers focus on preventing unauthorized access and ensuring secure data transmission. The Transport layer addresses end-to-end communication security through encryption and authentication mechanisms. The Session, Presentation, and Application layers deal with application-level security, including secure data presentation, user authentication, and application-specific vulnerabilities.

Security issues in computer systems often span multiple layers, and a comprehensive approach is essential to ensure robust protection. Common security concerns include unauthorized access, data interception, tampering, and denial-of-service attacks. Implementing encryption protocols, firewalls, intrusion detection systems, and secure coding practices are vital strategies to mitigate these risks. Additionally, ongoing security awareness, regular updates, and adherence to best practices contribute to building a resilient defense against evolving cyber threats. As technology advances, the OSI model remains a valuable tool for analyzing and addressing security issues in a systematic and layered manner.

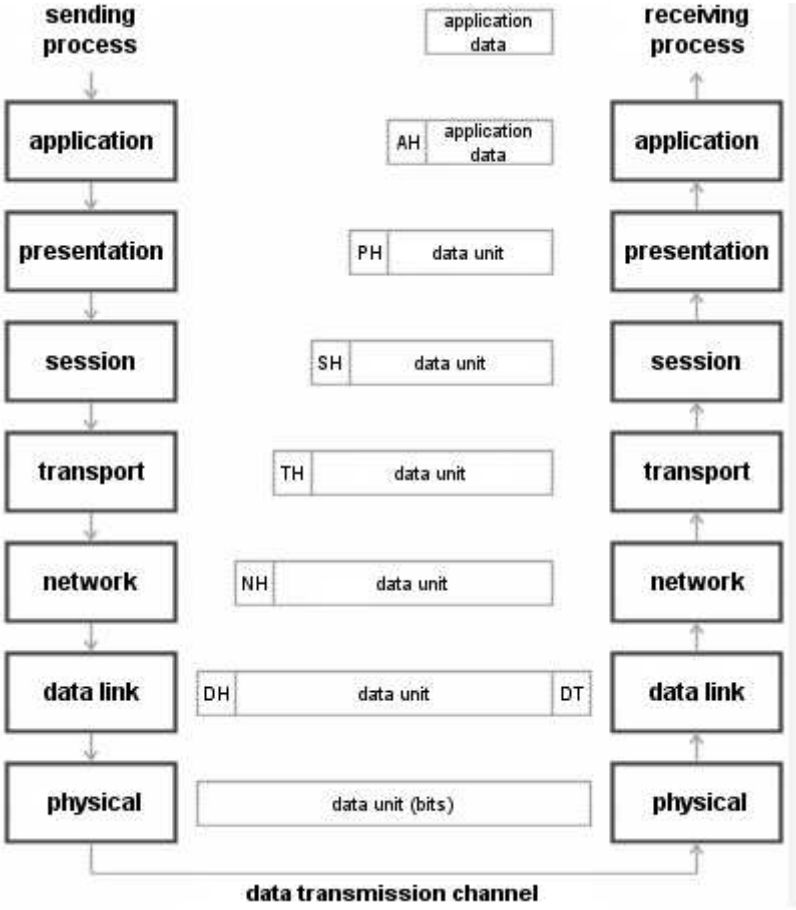

Figure 9: OSI Model

Seven layers of the OSI model

Physical layer

The physical layer refers to the physical communication medium and the technologies to transmit data across that medium. At its core, data communication is the transfer of digital and electronic signals through various physical channels like fiber-optic cables, copper cabling, and air. The physical layer includes standards for technologies and metrics closely related with the channels, such as Bluetooth, NFC, and data transmission speeds.

Data link layer

The data link layer refers to the technologies used to connect two machines across a network where the physical layer already exists. It manages data frames, which are digital signals encapsulated into data packets. Flow control and error control of data are often key focuses of the data link layer. Ethernet is an example of a standard at this level. The data link layer is often split into two sub-layers: the Media Access Control (MAC) layer and Logical Link Control (LLC) layer.

Network layer

The network layer is concerned with concepts such as routing, forwarding, and addressing across a dispersed network or multiple connected networks of nodes or machines. The network layer may also

manage flow control. Across the internet, the Internet Protocol v4 (IPv4) and IPv6 are used as the main network layer protocols.

Transport layer

The primary focus of the transport layer is to ensure that data packets arrive in the right order, without losses or errors, or can be seamlessly recovered if required. Flow control, along with error control, is often a focus at the transport layer. At this layer, commonly used protocols include the Transmission Control Protocol (TCP), a near-lossless connection-based protocol, and the User Datagram Protocol (UDP), a lossy connectionless protocol. TCP is commonly used where all data must be intact (e.g. file share), whereas UDP is used when retaining all packets is less critical (e.g. video streaming).

Session layer

The session layer is responsible for network coordination between two separate applications in a session. A session manages the beginning and ending of a one-to-one application connection and synchronization conflicts. Network File System (NFS) and Server Message Block (SMB) are commonly used protocols at the session layer.

Presentation layer

The presentation layer is primarily concerned with the syntax of the data itself for applications to send and consume. For example, Hypertext Markup Language (HTML), JavaScipt Object Notation (JSON), and Comma Separated Values (CSV) are all modeling languages to describe the structure of data at the presentation layer.

Application layer

The application layer is concerned with the specific type of application itself and its standardized communication methods. For example, browsers can communicate using HyperText Transfer Protocol Secure (HTTPS), and HTTP and email clients can communicate using POP3 (Post Office Protocol version 3) and SMTP (Simple Mail Transfer Protocol).

EXPERIMENT-12

Aim:- Creating and manipulation of Database table using SQL in MS Access.

Creating and manipulating database tables using SQL in Microsoft Access involves using SQL statements.

Creating a Table

To create a table, you can use the CREATE TABLE statement.

```
CREATE TABLE TableName (
    Column1 datatype,
    Column2 datatype,
    ...
    ColumnN datatype
);
```

Example

```
CREATE TABLE Employees (
    EmployeeID INT PRIMARY KEY,
    FirstName VARCHAR(50),
    LastName VARCHAR(50),
    BirthDate DATE,
    Salary DECIMAL(10, 2)
);
```

Adding Data to a Table

To insert data into a table, you use the INSERT INTO statement:

```
INSERT INTO TableName (Column1, Column2, ..., ColumnN)
VALUES (Value1, Value2, ..., ValueN);
```

Example:

```
INSERT INTO Employees (EmployeeID, FirstName, LastName, BirthDate, Salary)
VALUES (1, 'John', 'Doe', '1990-01-01', 50000.00);
```

Updating Data in a Table:

To update existing data in a table, use the UPDATE statement:

```
UPDATE TableName
SET Column1 = Value1, Column2 = Value2, ...
WHERE Condition;
```

Example:

```
UPDATE Employees
SET Salary = 55000.00
WHERE EmployeeID = 1;
```

Deleting Data from a Table:

To delete data from a table, use the DELETE FROM statement:

```
DELETE FROM TableName
WHERE Condition;
```

Example:

```
DELETE FROM Employees
WHERE EmployeeID = 1;
```

SOME ADDITIONAL PROGRAMS

WAP to illustrate Function overloading

Function overloading:- Function overloading is a feature of object-oriented programming where two or more functions can have the same name but different parameters. When a function name is overloaded with different jobs it is called Function Overloading. In Function Overloading "Function" name should be the same and the arguments should be different. Function overloading can be considered as an example of a polymorphism feature in C++.

If multiple functions having same name but parameters of the functions should be different is known as Function Overloading.

If we have to perform only one operation and having same name of the functions increases the readability of the program.

Suppose you have to perform addition of the given numbers but there can be any number of arguments, if you write the function such as a(int,int) for two parameters, and b(int,int,int) for three parameters then it may be difficult for you to understand the behavior of the function because its name differs.

Program

```
#include <iostream>
class Overloader {
public:
    // Function to add two integers
    int add(int a, int b) {
        return a + b;
    }

    // Function to add three integers
    int add(int a, int b, int c) {
        return a + b + c;
    }

    // Function to concatenate two strings
    std::string concatenate(std::string str1, std::string str2) {
        return str1 + str2;
    }
```

```
    // Function to add two double numbers
    double add(double x, double y) {
        return x + y;
    }
};

int main() {
    Overloader overloader;

    // Testing the different overloaded functions
    int sum1 = overloader.add(5, 10);
    int sum2 = overloader.add(5, 10, 15);

    std::string resultStr = overloader.concatenate("Hello, ", "world!");

    double sumDouble = overloader.add(3.5, 2.5);

    // Displaying the results
    std::cout << "Sum (int): " << sum1 << std::endl;
    std::cout << "Sum (int): " << sum2 << std::endl;
    std::cout << "Concatenation (string): " << resultStr << std::endl;
    std::cout << "Sum (double): " << sumDouble << std::endl;

    return 0;
}
```

Output

Sum (int): 15

Sum (int): 30

Concatenation (string): Hello, world!

Sum (double): 6

In this example

The Overloader class has multiple functions named add and concatenate with different parameter lists

The add function is overloaded for integers and doubles, and the concatenate function is overloaded for strings

In the main function, the different overloaded functions are called and their results are displayed.

Viva Voce Questions-

- What is function overloading?
- What are the advantages of function overloading?
- What happens if there are two overloaded functions with the same signature?
- How does function overloading differ from function overriding in C++?

WAP to illustrate Derived classes & Inheritance

Inheritance:- Inheritance is a feature or a process in which, new classes are created from the existing classes. The new class created is called “derived class” or “child class” and the existing class is known as the “base class” or “parent class”. The derived class now is said to be inherited from the base class.

When we say derived class inherits the base class, it means, the derived class inherits all the properties of the base class, without changing the properties of base class and may add new features to its own. These new features in the derived class will not affect the base class. The derived class is the specialized class for the base class.

- Sub Class: The class that inherits properties from another class is called Subclass or Derived Class.
- Super Class: The class whose properties are inherited by a subclass is called Base Class or Superclass

Program

```
#include <iostream>
#include <string>

// Base class
class Animal {
protected:
    std::string name;

public:
    // Constructor
    Animal(const std::string& n) : name(n) {}

    // Member function
    void eat() const {
        std::cout << name << " is eating." << std::endl;
    }

    // Virtual function to be overridden by derived classes
    virtual void sound() const {
```

```
        std::cout << "Animal makes a sound." << std::endl;
    }
};

// Derived class 1
class Dog : public Animal {
public:
    // Constructor
    Dog(const std::string& n) : Animal(n) {}

    // Override the sound function
    void sound() const override {
        std::cout << name << " barks: Woof! Woof!" << std::endl;
    }

    // New member function specific to Dog
    void fetch() const {
        std::cout << name << " is fetching the ball." << std::endl;
    }
};

// Derived class 2
class Cat : public Animal {
public:
    // Constructor
    Cat(const std::string& n) : Animal(n) {}

    // Override the sound function
    void sound() const override {
        std::cout << name << " meows: Meow! Meow!" << std::endl;
    }

    // New member function specific to Cat
```

```
    void climb() const {
        std::cout << name << " is climbing a tree." << std::endl;
    }
};

int main() {
    // Creating objects of derived classes
    Dog myDog("Buddy");
    Cat myCat("Whiskers");

    // Calling member functions of the base class
    myDog.eat();
    myDog.sound();

    myCat.eat();
    myCat.sound();

    // Calling member functions specific to derived classes
    myDog.fetch();
    myCat.climb();

    return 0;
}
```

Output

Buddy is eating.

Buddy barks: Woof! Woof!

Whiskers is eating.

Whiskers meows: Meow! Meow!

Buddy is fetching the ball.

Whiskers is climbing a tree.

In this example

Animal is the base class with a common eat function and a virtual function sound

Dog and Cat are derived classes that inherit from the Animal base class.

The sound function is overridden in both Dog and Cat classes.

Each derived class also has a new member function (fetch for Dog and climb for Cat).

Viva Voce Questions-

- What is inheritance in C++
- What are the types of inheritance in C++
- What is the access specifier used for inheritance in C++
- Can you provide an example of a real-world scenario where inheritance is useful?

WAP to insert and delete and element from the Stack

Stack is a linear data structure that follows a particular order in which the operations are performed. The order may be LIFO (Last In First Out) or FILO(First In Last Out). LIFO implies that the element that is inserted last, comes out first and FILO implies that the element that is inserted first, comes out last.

Program

```
#include <iostream>
#include <stack>

int main() {
    // Create an empty stack
    std::stack<int> myStack;

    // Insert elements into the stack (push)
    myStack.push(10);
    myStack.push(20);
    myStack.push(30);

    // Display the elements in the stack
    std::cout << "Elements in the stack: ";
    while (!myStack.empty()) {
        std::cout << myStack.top() << " ";
        myStack.pop();  // Remove the top element
    }
    std::cout << std::endl;

    // Reinsert elements for further demonstration
    myStack.push(40);
    myStack.push(50);

    // Display the top element without removing it
    if (!myStack.empty()) {
```

```
        std::cout << "Top element in the stack: " << myStack.top() << std::endl;
    } else {
        std::cout << "The stack is empty." << std::endl;
    }

    // Remove the top element from the stack (pop)
    if (!myStack.empty()) {
        myStack.pop();
        std::cout << "Top element removed from the stack." << std::endl;
    } else {
        std::cout << "Cannot pop from an empty stack." << std::endl;
    }

    // Display the updated elements in the stack
    std::cout << "Updated elements in the stack: ";
    while (!myStack.empty()) {
        std::cout << myStack.top() << " ";
        myStack.pop();
    }
    std::cout << std::endl;

    return 0;
}
```

Output

Elements in the stack: 30 20 10

Top element in the stack: 50

Top element removed from the stack.

Updated elements in the stack: 40

In this program:

- The <stack> header is included to use the std::stack container.
- Elements are inserted into the stack using the push operation.
- The top function is used to access the top element without removing it.
- The pop operation is used to remove the top element from the stack.

- The program checks for empty stack conditions before attempting to access or remove elements.

Viva Voce Questions-

- What is a stack?
- How can you implement a stack in C++?
- How is dynamic memory allocation related to the stack?
- What is stack overflow?

WAP to insert and delete and element from the Queue

Queue:- A Queue is defined as a linear data structure that is open at both ends and the operations are performed in First In First Out (FIFO) order. We define a queue to be a list in which all additions to the list are made at one end, and all deletions from the list are made at the other end.

Program

```
#include <iostream>
#include <queue>
int main() {
    // Create an empty queue
    std::queue<int> myQueue;

    // Insert elements into the queue (enqueue)
    myQueue.push(10);
    myQueue.push(20);
    myQueue.push(30);

    // Display the elements in the queue
    std::cout << "Elements in the queue: ";
    while (!myQueue.empty()) {
        std::cout << myQueue.front() << " ";
        myQueue.pop();  // Remove the front element
    }
    std::cout << std::endl;

    // Reinsert elements for further demonstration
    myQueue.push(40);
    myQueue.push(50);

    // Display the front element without removing it
    if (!myQueue.empty()) {
        std::cout << "Front element in the queue: " << myQueue.front() << std::endl;
    } else {
```

```
        std::cout << "The queue is empty." << std::endl;
    }

    // Remove the front element from the queue (dequeue)
    if (!myQueue.empty()) {
        myQueue.pop();
        std::cout << "Front element removed from the queue." << std::endl;
    } else {
        std::cout << "Cannot dequeue from an empty queue." << std::endl;
    }

    // Display the updated elements in the queue
    std::cout << "Updated elements in the queue: ";
    while (!myQueue.empty()) {
        std::cout << myQueue.front() << " ";
        myQueue.pop();
    }
    std::cout << std::endl;

    return 0;
}
```

Output

Elements in the queue: 10 20 30

Front element in the queue: 40

Front element removed from the queue.

Updated elements in the queue: 50

In this program

The <queue> header is included to use the std::queue container.

Elements are inserted into the queue using the push operation.

The front function is used to access the front element without removing it.

The pop operation is used to remove the front element from the queue.

The program checks for empty queue conditions before attempting to access or remove elements.

Viva Voce Questions-

- Why is the queue data structure useful in programming?
- How does a priority queue differ from a regular queue?
- Explain the concept of the "front" and "rear" in a queue?
- How can you implement a queue in C++?

WAP to insert and delete and element from the Linked List

Linked List:- A linked list is the most sought-after data structure when it comes to handling dynamic data elements. A linked list consists of a data element known as a node. And each node consists of two fields: one field has data, and in the second field, the node has an address that keeps a reference to the next node.

Program

```
#include<bits/stdc++.h>
using namespace std;
struct Node{
   int data;
   struct Node *next;
};
void Insert(struct Node **head,int position,int x)
{
   //cout<<(head)<<" "<<(*head)<<"\n";
   struct Node *p,*q,*newNode;
   newNode=new Node();
   newNode->data=x;
   p=*head;
   if(position==1)
   {
      newNode->next=p;
      //cout<<newNode<<endl;
      *head=newNode;
      return;
   }
   else
   {
      int k=1;
      while(p!=NULL && k<position)
```

```
        {
            k++;
            q=p;
            p=p->next;
        }
        q->next=newNode;
        newNode->next=p;
    }
}
void Delete(struct Node **head,int position)
{
    struct Node *p,*q;
    p=*head;
    //cout<<(head)<<" "<<(*head)<<"\n";
    if(*head==NULL)
    {
        cout<<"List empty!!\n";
        return;
    }

    if(position==1)
    {
        *head=(*head)->next;
        free(p);
        return;
    }
    else
    {
        int k=1;
        while(p!=NULL && k<position)
        {
            k++;
            q=p;
```

```
            p=p->next;
        }
        if(p==NULL)
        {
            cout<<"Position not found!!\n";
            return;
        }
        else
        {
            q->next=p->next;
            delete(p);
        }
    }
}
void Print(struct Node **head)
{
    struct Node *p=*head;
    while(p)
    {
        cout<<p->data<<" ";
        p=p->next;
    }
    cout<<"\n";
}
int main()
{
    struct Node *head=NULL;
    //cout<<head<<"\n";
    Insert(&head,1,5);
    Insert(&head,1,4);
    Insert(&head,3,3);
    Insert(&head,2,2);
    Insert(&head,2,1);
```

```
        Print(&head);

        Delete(&head,1);
        Print(&head);
        Delete(&head,4);
        //Delete(&head,1);
        Print(&head);
        Delete(&head,4);
        Print(&head);
    }
```

Output

4 1 2 5 3

1 2 5 3

1 2 5

Position not found!!

1 2 5

Viva Voce Questions-

- What is a Linked List?
- What is a Singly Linked List?
- What is the difference between an array and a linked list?
- What is a circular linked list?

Write a C++ program to check whether a given number is even or odd

Program

```
#include <iostream>

int main() {
   int number;

   // Input
   std::cout << "Enter a number: ";
   std::cin >> number;

   // Check whether the number is even or odd
   if (number % 2 == 0) {
      std::cout << number << " is even." << std::endl;
   } else {
      std::cout << number << " is odd." << std::endl;
   }

   return 0;
}
```

Output

```
Enter a number: 2
2 is even.
Enter a number: 5
5    is odd.
```

Write a C++ program to find whether a given year is a leap year or not

Program

```
#include <iostream>

int main() {
    int year;

    // Input
    std::cout << "Enter a year: ";
    std::cin >> year;

    // Check if the year is a leap year
    if ((year % 4 == 0 && year % 100 != 0) || (year % 400 == 0)) {
        std::cout << year << " is a leap year." << std::endl;
    } else {
        std::cout << year << " is not a leap year." << std::endl;
    }

    return 0;
}
```

Output

Enter a year: 2001

2001 is not a leap year

Enter a year: 2024

2024 is a leap year

Viva Voce Questions-

- **What is C++?**

C++ is a general-purpose programming language that is an extension of the C programming language. It supports object-oriented programming features as well.

- **What is the difference between C and C++?**

C++ is an extension of C with additional features, including classes and objects for object-oriented programming, function overloading, and templates.

- **What is Object-Oriented Programming (OOP)?**

Object-Oriented Programming is a programming paradigm that uses objects (instances of classes) to

organize code. It promotes concepts such as encapsulation, inheritance, and polymorphism.

- **What are the basic principles of OOP?**

Encapsulation, Inheritance, and Polymorphism, Inheritance, classes and objects.

- **Explain the concept of a class in C++.**

A class is a blueprint for creating objects. It defines a data structure along with the methods that operate on that data.

- **What is an object in C++?**

An object is an instance of a class. It represents a real-world entity and is created from a class.

What is the difference between public, private, and protected access specifiers in a class?

These are access control keywords in C++. Public members are accessible from outside the class, private members are only accessible within the class, and protected members are accessible within the class and its derived classes.

- **What is function overloading?**

Function overloading is the ability to define multiple functions in the same scope with the same name but different parameters.

- **Explain the concept of inheritance in C++.**

Inheritance allows a class to inherit properties and behaviors from another class. It supports the creation of a new class (derived class) using an existing class (base class) as a foundation.

- **What is a constructor in C++?**

A constructor is a special member function with the same name as the class. It is automatically called when an object is created and is used to initialize the object's state.

- **What is a destructor in C++?**

A destructor is a special member function with the same name as the class, preceded by a tilde (~). It is automatically called when an object goes out of scope and is used to release resources or perform cleanup.

What is the difference between new and malloc()?

new is an operator in C++ for dynamic memory allocation, while malloc() is a function in C. new automatically calls constructors, whereas malloc() does not.

- **What is the Standard Template Library (STL) in C++?**

The STL is a collection of template classes and functions in C++ that provides general-purpose classes and functions with templates that implement many popular and commonly used algorithms and data structures like vectors, lists, queues, and

more.

VISION OF THE DEPARTMENT

To be a centre of excellence for providing quality technical education to develop future leaders with the aspects of research &computing, Software product development and entrepreneurship.

MISSION OF THE DEPARTMENT

M1: To offer academic programme with state of art curriculum having flexibility for accommodating the latest developments in the areas of Computer science engineering

M2: To conduct research and development activities in contemporary and emerging areas of computer science & engineering.

M3: To inculcate moral values & entrepreneurial skills to produce professionals capable of providing socially relevant and sustainable solutions.