Mewar University

Chittorgarh

Lab Manual

Object Oriented Programming Lab

B.TECH. IV SEMESTER

Branch: Computer Science & Engineering

Session 2022-23

Department of Computer Science & Engineering

Mewar University, Chittorgarh

Chittorgarh, 312001

TABLE OF CONTENTS

LAB RULES

Responsibilities of Users

Users are expected to follow some fairly obvious rules of conduct:

Always:

- Enter the lab on time and leave at proper time.
- Wait for the previous class to leave before the next class enters.
- Keep the bag outside in the respective racks.
- Utilize lab hours in the corresponding.
- Turn off the machine before leaving the lab unless a member of lab staff has specifically told you not to do so.
- Leave the labs at least as nice as you found them.
- If you notice a problem with a piece of equipment (e.g. a computer doesn't respond) or the room in general (e.g. cooling, heating, lighting) please report it to lab staff immediately. Do not attempt to fix the problem yourself.

Never:

- Don't harm any equipment.
- Do not adjust the heat or air conditioners. If you feel the temperature is not properly set, inform lab staff; we will attempt to maintain a balance that is healthy for people and machines.
- Do not attempt to reboot a computer. Report problems to lab staff.
- Do not remove or modify any software or file without permission.

- Do not remove printers and machines from the network without being explicitly told to do so by lab staff.
- Don't monopolize equipment. If you're going to be away from your machine for more than 10 or 15 minutes, log out before leaving. This is both for the security of your account, and to ensure that others are able to use the lab resources while you are not.
- Don't use internet, internet chat of any kind in your regular lab schedule.
- Do not download or upload of MP3, JPG or MPEG files.
- No games are allowed in the lab sessions.
- No hardware including USB drives can be connected or disconnected in the labs without prior permission of the lab in-charge.
- No food or drink is allowed in the lab or near any of the equipment. Aside from the fact that it leaves a mess and attracts pests, spilling anything on a keyboard or other piece of computer equipment could cause permanent, irreparable, and costly damage. (and in fact *has*) If you need to eat or drink, take a break and do so in the canteen.
- Don't bring any external material in the lab, except your lab record, copy and books.
- Don't bring the mobile phones in the lab. If necessary, then keep them in silence mode.
- Please be considerate of those around you, especially in terms of noise level. While labs are a natural place for conversations of all types, kindly keep the volume turned down.

If you are having problems or questions, please go to either the faculty, lab in-charge or the lab supporting staff. They will help you. We need your full support and cooperation for smooth functioning of the lab.

INSTRUCTIONS

Before entering in the lab

All the students are supposed to prepare the theory regarding the next experiment.

Students are supposed to bring the practical file and the lab copy.

Previous programs should be written in the practical file.

All the students must follow the instructions, failing which he/she may not be allowed in the lab.

While working in the lab

Adhere to experimental schedule as instructed by the lab in-charge.

Get the previously executed program signed by the instructor.

Get the output of the current program checked by the instructor in the lab copy.

Each student should work on his/her assigned computer at each turn of the lab.

Take responsibility of valuable accessories.

Concentrate on the assigned practical and do not play games

If anyone caught red handed carrying any equipment of the lab, then he/she will have to face serious consequences.

LIST OF EXPERIMENTS

SR. No	Experiments
1	Write a C++ program to display Names, Roll No., and grade of 3 students who have appeared in the examination. Declare the class of name, roll no., and grade. Create an array of class objects. Read and display the contents of the array.
2	Write a C++ program to declare struct. Initialize and display contents of member
3	Write a C++ program to declare a class. Declare pointer to class. Initialize and display the contents of the class members
4	Given that EMPLOYEE class contains following members: data members: Employee number, Employee name, Basic, DA, IT, Net Salary, and print data members
5	Write a C++ program to read the data of N employee and compute the net salary of each employee (DA=52% of basic and IT=30% of gross salary).
6	Write a C++ to illustrate the concepts of console I/O operations
7	Write a C++ program to use scope resolution operator. Display the various values of the same variables declared at different scope levels.
8	Write a C++ program to allocate memory using new operator.
9	Write a C++ program to create multilevel inheritance
10	Write a C++ program to create an array of pointers. Invoke functions using array objects
11	Write a C++ program to use pointer for both base and derived classes and call the member function. Use virtual keyword

EXPERRIMENT -I

Creation of Student Class

```
// C Program to Insert an element
#include <iostream>
using namespace std;

class Student {
public:
    string name;
    int rollNo;
    char grade;

    // Function to input student details
    void inputDetails() {
        cout << "Enter name: ";
        cin >> name;
        cout << "Enter Roll No: ";
        cin >> rollNo;
        cout << "Enter Grade: ";
        cin >> grade;
    }

    // Function to display student details
    void displayDetails() {
        cout << "Name: " << name << ", Roll No: " << rollNo << ", Grade: " << grade << endl;
    }
};

int main() {
    const int SIZE = 3;
    Student students[SIZE]; // Array of Student objects

    // Input details for each student
    cout << "Enter details for " << SIZE << " students:" << endl;
    for(int i = 0; i < SIZE; i++) {
        cout << "Student " << (i+1) << ":" << endl;
        students[i].inputDetails();
    }

    // Display details for each student
    cout << "\nDisplaying Student Details:" << endl;
    for(int i = 0; i < SIZE; i++) {
        students[i].displayDetails();
    }

    return 0;
}
```

Output

Enter details for 3 students:
Enter name: Sunil
Enter Roll No: 12122
Enter Grade: A
Enter name: Rohit
Enter Roll No: 12324
Enter Grade: B
Enter name: Akshay
Enter Roll No: 12432
Enter Grade: C

Displaying Student Details:
Name: Sunil, Roll No: 12122, Grade: A
Name: Rohit, Roll No: 12324, Grade: B
Name: Akshay, Roll No: 12432, Grade: C

Declaration and Initialization of Structure

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

// Declaration of the struct
struct Person {
    string name;
    int age;
    char gender;
};

int main() {
    // Initialize an instance of Person
    Person person1 = {"John Doe", 30, 'M'};

    // Display the contents of person1
    cout << "Person Details:" << endl;
    cout << "Name: " << person1.name << endl;
    cout << "Age: " << person1.age << endl;
    cout << "Gender: " << person1.gender << endl;

    return 0;
}
```

Output

```
Person Details:
Name: John Doe
Age: 30
Gender: M
```

EXPERRIMENT -3
Declaration of Class with Pointer variable

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

// Declaration of the class
class Student {
public:
    string name;
    int age;

    // Constructor to initialize members
    Student(string n, int a) : name(n), age(a) {}

    // Function to display student details
    void display() {
        cout << "Student Details:" << endl;
        cout << "Name: " << name << endl;
        cout << "Age: " << age << endl;
    }
};

int main() {
    // Creating an instance of Student dynamically
    Student* studentPtr = new Student("Alice", 20);

    // Display the contents using the pointer
    studentPtr->display();

    // Freeing the allocated memory
    delete studentPtr;

    return 0;
}
```

Output:-

Student Details:
Name: Alice
Age: 20

EXPERRIMENT -4

Class creation of Employee with basic salary

```
#include <iostream>
using namespace std;
class EMPLOYEE {
private:
 int empNumber;
 string empName;
 float basic, da, it, netSalary;

public:
 // Constructor to initialize the data members
 EMPLOYEE(int number, string name, float basicSalary, float daRate, float itRate)
    : empNumber(number), empName(name), basic(basicSalary), da(daRate), it(itRate) {
    calculateNetSalary(); // Calculate Net Salary during object creation
 }
```

```
// Function to calculate Net Salary
void calculateNetSalary() {
    // Assuming DA is a percentage of Basic and IT is a deduction amount
    float daAmount = (basic * da) / 100.0;
    netSalary = basic + daAmount - it; // Net Salary calculation
}

// Function to print employee details
void printDetails() {
    cout << "Employee Number: " << empNumber << endl;
    cout << "Employee Name: " << empName << endl;
    cout << "Basic Salary: " << basic << endl;
    cout << "DA: " << da << "%" << endl;
    cout << "IT: " << it << endl;
    cout << "Net Salary: " << netSalary << endl;
}
};

int main() {
// Creating an EMPLOYEE object with example data
EMPLOYEE emp1(123, "John Doe", 50000.0, 10.0, 5000.0);

// Displaying the employee details
```

```
emp1.printDetails();


return 0;

}
```

Output

Employee Number: 123

Employee Name: John Doe

Basic Salary: 50000

DA: 10%

IT: 5000

Net Salary: 50000

EXPERIMENT -5

Computation of Employee salary with class members

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

class Employee {
public:
    int empNumber;
    string empName;
    float basic, da, it, grossSalary, netSalary;

    // Function to input employee details
    void inputDetails(int number) {
        empNumber = number;
        cout << "Enter Employee Name: ";
        cin >> empName;
        cout << "Enter Basic Salary: ";
```

```
        cin >> basic;
    }

    // Function to calculate DA, IT, and Net Salary
    void calculateSalaries() {
        da = basic * 0.52; // DA is 52% of basic
        grossSalary = basic + da;
        it = grossSalary * 0.30; // IT is 30% of gross salary
        netSalary = grossSalary - it;
    }

    // Function to print employee details
    void printDetails() {
        cout << "Employee Number: " << empNumber << endl;
        cout << "Employee Name: " << empName << endl;
        cout << "Basic Salary: $" << basic << endl;
        cout << "DA: $" << da << " (52% of Basic)" << endl;
        cout << "Gross Salary: $" << grossSalary << endl;
        cout << "IT: $" << it << " (30% of Gross)" << endl;
        cout << "Net Salary: $" << netSalary << endl;
        cout << "----------------------------------" << endl;
    }
};

int main() {
    int N;
    cout << "Enter the number of employees: ";
    cin >> N;

    vector<Employee> employees(N); // Create a vector of Employee objects

    // Input and calculate details for each employee
    for (int i = 0; i < N; i++) {
        cout << "Enter details for employee " << (i + 1) << ":" << endl;
        employees[i].inputDetails(i + 1);
        employees[i].calculateSalaries();
    }

    // Display details for each employee
    cout << "\nEmployee Details and Net Salaries:" << endl;
    for (int i = 0; i < N; i++) {
```

```
            employees[i].printDetails();
        }

        return 0;
    }
```

Output

Enter the number of employees: 2
Enter details for employee 1:
Enter Employee Name: Akshay
Enter Basic Salary: 70000
Enter details for employee 2:
Enter Employee Name: Rohit
Enter Basic Salary: 50000

Employee Details and Net Salaries:
Employee Number: 1
Employee Name: Akshay
Basic Salary: $70000
DA: $36400 (52% of Basic)
Gross Salary: $106400
IT: $31920 (30% of Gross)
Net Salary: $74480

Employee Number: 2
Employee Name: Rohit
Basic Salary: $50000
DA: $26000 (52% of Basic)
Gross Salary: $76000
IT: $22800 (30% of Gross)
Net Salary: $53200

EXPERIMENT -6

I/O operation implementation with class

```
#include <iostream>
#include <string> // Include for using the string class

using namespace std;

int main() {
    // Variables to hold user input
    string name;
    int age;

    // Output operations using cout
    cout << "Enter your name: ";
    // Input operation using getline to read a line of text
    getline(cin, name);

    cout << "Enter your age: ";
    // Input operation using cin
    cin >> age;

    // More output operations using cout
    cout << "Hello, " << name << "!" << endl;
    cout << "You are " << age << " years old." << endl;

    // Illustrating formatted output
    double pi = 3.141592653589793;
    cout << "Unformatted PI: " << pi << endl;
    // Set precision for floating-point output
    cout.precision(3);
    cout << "Formatted PI (3 decimal places): " << pi << endl;

    // Demonstrating error output using cerr
    cerr << "This is an error message example using cerr" << endl;
```

```
    // Demonstrating logging output using clog
    clog << "This is a log message example using clog" << endl;

    return 0;
}
```

Output:

Enter your name: Akshay

Enter your age: 27

Hello, Akshay!

You are 27 years old.

Unformatted PI: 3.14159

Formatted PI (3 decimal places): 3.14

This is an error message example using cerr

This is a log message example using clog

EXPERIMENT -7

Scope Resolution Operator in C++

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

int value = 10; // Global variable declaration

class Demo {
public:
    static int value; // Static class member variable declaration
    void display() const {
        int value = 30; // Local variable declaration inside a method
        cout << "Local value in display(): " << value << endl;
        cout << "Static value in Demo class: " << Demo::value << endl;
        cout << "Global value: " << ::value << endl;
    }
};

int Demo::value = 20; // Definition of static class member variable

int main() {
    Demo demo;
    demo.display();

    int value = 40; // Local variable in main()
    cout << "Local value in main(): " << value << endl;
    cout << "Static value in Demo class accessed in main(): " << Demo::value << endl;
    cout << "Global value accessed in main(): " << ::value << endl;

    return 0;
}
```

Output:

Local value in display(): 30

Static value in Demo class: 20

Global value: 10

Local value in main(): 40

Static value in Demo class accessed in main(): 20

Global value accessed in main(): 10

EXPERIMENT -8

Dynamic Memory Allocation with new Operator

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

class Sample {
public:
    Sample() { cout << "Sample object created.\n"; }
    ~Sample() { cout << "Sample object destroyed.\n"; }
};

int main() {
    // Allocate memory for a single integer
    int* ptr = new int(5); // Initializes the memory with 5
    cout << "Value of *ptr: " << *ptr << endl;

    // Deallocate memory for the integer
    delete ptr;
    ptr = nullptr; // Good practice to assign nullptr after deleting

    // Allocate memory for an object of Sample
    Sample* samplePtr = new Sample();

    // No need to explicitly call the constructor, new does that

    // Deallocate memory for the object
    delete samplePtr;
    samplePtr = nullptr;

    // Allocate memory for an array of integers
    int* arrayPtr = new int[3] {1, 2, 3}; // Initializes the array

    cout << "Array elements: ";
    for (int i = 0; i < 3; ++i) {
        cout << arrayPtr[i] << " ";
    }
    cout << endl;

    // Deallocate memory for the array
    delete[] arrayPtr;
    arrayPtr = nullptr;

    return 0;
}
```

OUTPUT

```
Value of *ptr: 5
Sample object created.
Sample object destroyed.
Array elements: 1 2    3
```

EXPERIMENT -9

Multilevel Inheritance

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

// Base class
class BaseClass {
public:
    void displayBase() {
        cout << "Displaying Base class content." << endl;
    }
};

// Intermediate class derived from BaseClass
class IntermediateClass : public BaseClass {
public:
    void displayIntermediate() {
        cout << "Displaying Intermediate class content." << endl;
    }
};

// Derived class derived from IntermediateClass
class DerivedClass : public IntermediateClass {
public:
    void displayDerived() {
        cout << "Displaying Derived class content." << endl;
    }
};

int main() {
    DerivedClass obj;

    // Object of DerivedClass can access members of all the classes in the hierarchy
    obj.displayBase();          // Accessing Base class member
    obj.displayIntermediate();  // Accessing Intermediate class member
    obj.displayDerived();       // Accessing Derived class member

    return 0;
}
```

OUTPUT

Displaying Base class content.
Displaying Intermediate class content.
Displaying Derived class content

EXPERIMENT 10

Invoke functions using array objects

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

class Sample {
public:
    int id;

    Sample(int id): id(id) {}  // Constructor with an initializer list

    void display() {
        cout << "Displaying Sample object with ID: " << id << endl;
    }
};

int main() {
    const int size = 3;  // Size of the array of pointers
    Sample* sampleArray[size];  // Array of pointers to Sample objects

    // Dynamically allocate the Sample objects and store their pointers in the array
    for(int i = 0; i < size; ++i) {
        sampleArray[i] = new Sample(i + 1);  // Object IDs will be 1, 2, 3
    }

    // Invoke the display function using each object in the array
    for(int i = 0; i < size; ++i) {
        sampleArray[i]->display();
    }

    // Don't forget to delete the dynamically allocated objects to avoid memory leaks
    for(int i = 0; i < size; ++i) {
        delete sampleArray[i];
    }

    return 0;
}
```

INPUT/ OUTPUT

```
Displaying Sample object with ID: 1
Displaying Sample object with ID: 2
Displaying Sample object with ID: 3
```

EXPERIMENT- 11

Calling the function Using Virtual keyword

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

// Base class
class Base {
public:
    virtual void show() {  // Virtual function
        cout << "Base class show" << endl;
    }

    // Including a virtual destructor ensures that derived class destructors are called correctly
    virtual ~Base() {
        cout << "Base destructor" << endl;
    }
};

// Derived class
class Derived : public Base {
public:
    void show() override {  // Override the base class function
        cout << "Derived class show" << endl;
    }

    ~Derived() {
        cout << "Derived destructor" << endl;
    }
};

int main() {
    Base *basePtr;      // Base class pointer
    Derived derivedObj;  // Derived class object

    basePtr = &derivedObj;  // Pointing base class pointer to derived class object

    // Calls the function according to the type of object pointed to by basePtr, i.e., Derived
class's show()
```

```
    basePtr->show();

    // Because Base's destructor is virtual, deleting basePtr will correctly call Derived's destructor first, then Base's destructor
    // If you dynamically allocate memory, make sure to delete it to prevent memory leaks. For this static allocation, it's not needed.
    // delete basePtr;

    return 0;
```

OUTPUT

Derived class show
Derived destructor
Base destructor