Name:Dishan T. Shaikh

PRN:124B1C037

DIV-G2

SUB-OOPS.

1. Assume that a bank maintains two kinds of accounts for customers, one called as savings and the other as current account. The savings account provides compound interest and withdrawal facilities but no chequebook facility. The current account provides cheque book facility but no interest. Current
account holders should also maintain a minimum balance and if the balance falls below this level a service charge is imposed. Create a class account that stores customer name, account number and type
of account. From this derive the classes cur_acct and sav_acct to make them more specific to their requirements. Include necessary member functions in order to achieve the following tasks:

(a) Accept the deposit from a customer and update the balance.

(b) Display the balance.

(c) Compute and deposit interest.

(d) Permit withdrawal and update the balance.

(e) Check for the minimum balance, impose penalty, necessary and update the balance.

Do not use any constructors. Use member functions to initialize class members.

```
⍰#include <iostream>
using namespace std;
class Account {
protected:
char customerName[50];
int accountNumber;
char accountType[10];
double balance;
public
```

```
void initialize() {
    cout << "Enter customer name: ";
    cin.ignore(); // clear input buffer before getline-style input
    cin.getline(customerName, 50);

    cout << "Enter account number: ";
    cin >> accountNumber;

    cout << "Enter account type (Savings/Current): ";
    cin >> accountType;

    cout << "Enter initial balance: ";
    cin >> balance;
}

void deposit() {
    double amount;
    cout << "Enter amount to deposit: ";
    cin >> amount;
    if (amount > 0) {
        balance += amount;
        cout << "Deposited: " << amount << endl;
    } else {
        cout << "Invalid deposit amount.\n";
    }
}

void displayBalance() const {
    cout << "Current Balance: " << balance << endl;
}
```

```
    double getBalance() const {
        return balance;
    }

    void withdraw(double amount) {
        if (amount > 0 && amount <= balance) {
            balance -= amount;
            cout << "Withdrawn: " << amount << endl;
        } else {
            cout << "Invalid withdrawal amount or insufficient balance.\n";
        }
    }
};

// Savings Account
class SavAcct : public Account {
public:
    void computeAndDepositInterest() {
        double rate;
        int years;
        cout << "Enter interest rate (%): ";
        cin >> rate;
        cout << "Enter number of years: ";
        cin >> years;

        double interest = balance;
        for (int i = 0; i < years; ++i) {
            interest += (interest * rate / 100.0);
        }
        double earned = interest - balance;
        balance = interest;
        cout << "Interest of " << earned << " added. New balance: " << balance << endl;
    }

    void withdraw() {
        double amount;
        cout << "Enter amount to withdraw: ";
        cin >> amount;
        Account::withdraw(amount);
    }
};

// Current Account
class CurAcct : public Account {
    const double minBalance = 500.0;
    const double serviceCharge = 50.0;

public:
    void checkMinBalance() {
        if (balance < minBalance) {
            balance -= serviceCharge;
            cout << "Balance below minimum. Service charge of " << serviceCharge << " imposed.\n";
        } else {
            cout << "Minimum balance maintained.\n";
        }
    }

    void withdraw() {
        double amount;
        cout << "Enter amount to withdraw: ";
        cin >> amount;
        if (amount > 0 && amount <= balance) {
            balance -= amount;
            cout << "Withdrawn: " << amount << endl;
            checkMinBalance();
        } else {
            cout << "Invalid withdrawal amount or insufficient balance.\n";
        }
    }
};

int main() {
    char type;
    cout << "Select account type (S for Savings, C for Current): ";
    cin >> type;

    if (type == 'S' || type == 's') {
        SavAcct savings;
        savings.initialize();
        savings.deposit();
        savings.computeAndDepositInterest();
        savings.withdraw();
        savings.displayBalance();
    } else if (type == 'C' || type == 'c') {
        CurAcct current;
        current.initialize();
        current.deposit();
        current.withdraw();
        current.displayBalance();
    } else {
        cout << "Invalid account type selected.\n";
    }
    return 0;
}
```

OUTPUT:-

```
Select account type (S for Savings, C for Current): s
Enter customer name: aryan
Enter account number: 011
Enter account type (Savings/Current): Savings
Enter initial balance: 10000
Enter amount to deposit: 3000
Deposited: 3000
Enter interest rate (%): 20
Enter number of years: 3
Interest of 9464 added. New balance: 22464
Enter amount to withdraw: 200
Withdrawn: 200
Current Balance: 22264



=== Code Execution Successful ===
```

2. Modify the program of exercise 1 to include constructors for all three classes.

→

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

class Account {
protected:
    char customerName[50];
    int accountNumber;
    char accountType[10];
    double balance;

public:
    Account() {
        cout << "Enter customer name: ";
        cin.ignore(); // clear buffer
        cin.getline(customerName, 50);

        cout << "Enter account number: ";
        cin >> accountNumber;

        cout << "Enter account type (Savings/Current): ";
        cin >> accountType;

        cout << "Enter initial balance: ";
        cin >> balance;
    }

    void deposit() {
        double amount;
        cout << "Enter amount to deposit: ";
        cin >> amount;
        if (amount > 0) {
            balance += amount;
            cout << "Deposited: " << amount << endl;
        } else {
            cout << "Invalid deposit amount.\n";
        }
    }

    void displayBalance() const {
        cout << "Current Balance: " << balance << endl;
    }

    double getBalance() const {
        return balance;
    }

    void withdraw(double amount) {
        if (amount > 0 && amount <= balance) {
            balance -= amount;
            cout << "Withdrawn: " << amount << endl;
        } else {
            cout << "Invalid withdrawal amount or insufficient balance.\n";
        }
    }
};

// Savings Account
class SavAcct : public Account {
public:
    SavAcct() : Account() {
        // Constructor of base class handles initialization
    }

    void computeAndDepositInterest() {
        double rate;
        int years;
        cout << "Enter interest rate (%): ";
        cin >> rate;
        cout << "Enter number of years: ";
        cin >> years;
        double interest = balance;
        for (int i = 0; i < years; ++i) {
            interest += (interest * rate / 100.0);
        }
        double earned = interest - balance;
        balance = interest;

        cout << "Interest of " << earned << " added. New balance: " << balance << endl;
    }

    void withdraw() {
        double amount;
        cout << "Enter amount to withdraw: ";
        cin >> amount;
        Account::withdraw(amount);
    }
};

// Current Account
class CurAcct : public Account {
    const double minBalance = 500.0;
    const double serviceCharge = 50.0;

public:
    CurAcct() : Account() {
        // Constructor of base class handles initialization
    }

    void checkMinBalance() {
        if (balance < minBalance) {
            balance -= serviceCharge;
            cout << "Balance below minimum. Service charge of " << serviceCharge << " imposed.\n";
        } else {
            cout << "Minimum balance maintained.\n";
        }
    }

    void withdraw() {
        double amount;
        cout << "Enter amount to withdraw: ";
        cin >> amount;
        if (amount > 0 && amount <= balance) {
            balance -= amount;
            cout << "Withdrawn: " << amount << endl;
            checkMinBalance();
        } else {
            cout << "Invalid withdrawal amount or insufficient balance.\n";
        }
    }
};

int main() {
    char type;
    cout << "Select account type (S for Savings, C for Current): ";
    cin >> type;

    if (type == 'S' || type == 's') {
        SavAcct savings;
        savings.deposit();
        savings.computeAndDepositInterest();
        savings.withdraw();
        savings.displayBalance();
    } else if (type == 'C' || type == 'c') {
        CurAcct current;
        current.deposit();
        current.withdraw();
        current.displayBalance();
    } else {
        cout << "Invalid account type selected.\n";
    }

    return 0;
}
```

OUTPUT:-

```
Select account type (S for Savings, C for Current): c
Enter customer name: wini
Enter account number: 13
Enter account type (Savings/Current):
Savings
Enter initial balance: 1000
Enter amount to deposit: 200
Deposited: 200
Enter amount to withdraw: 100
Withdrawn: 100
Minimum balance maintained.
Current Balance: 1100


=== Code Execution Successful ===
```

3. Inheritance

An educational institution wishes to maintain a database of its employees. The database is divided into

several classes whose hierarchical relationships are shown in following figure. The figure also shows

the minimum information required for each class. Specify all classes and define functions to create the

database and retrieve individual details as and when needed.

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

class Employee {
protected:
   int empID;
   char name[50];
   char address[100];

public:
   void getEmployeeDetails() {
      cout << "Enter Employee ID: ";
      cin >> empID;
      cin.ignore();  // flush newline
      cout << "Enter Name: ";
      cin.getline(name, 50);
      cout << "Enter Address: ";
      cin.getline(address, 100);
   }

   void showEmployeeDetails() const {
      cout << "Employee ID: " << empID << endl;
      cout << "Name: " << name << endl;
      cout << "Address: " << address << endl;
   }
};
```

```
class Teaching : public Employee {
protected:
   char subject[30];
   char qualification[50];

public:
   void getTeachingDetails() {
      getEmployeeDetails();
      cout << "Enter Subject: ";
      cin.getline(subject, 30);
      cout << "Enter Qualification: ";
      cin.getline(qualification, 50);
   }

   void showTeachingDetails() const {
      showEmployeeDetails();
      cout << "Subject: " << subject << endl;
      cout << "Qualification: " << qualification << endl;
   }
};

class Professor : public Teaching {
   char designation[30];
   char researchArea[50];

public:
   void getProfessorDetails() {
      getTeachingDetails();
      cout << "Enter Designation: ";
      cin.getline(designation, 30);
      cout << "Enter Research Area: ";
```

```
        cin.getline(researchArea, 50);
    }

    void showProfessorDetails() const {
        showTeachingDetails();
        cout << "Designation: " << designation << endl;
        cout << "Research Area: " << researchArea << endl;
    }
};

class NonTeaching : public Employee {
    char department[30];
    char duty[50];

public:
    void getNonTeachingDetails() {
        getEmployeeDetails();
        cout << "Enter Department: ";
        cin.getline(department, 30);
        cout << "Enter Duty: ";
        cin.getline(duty, 50);
    }

    void showNonTeachingDetails() const {
        showEmployeeDetails();
        cout << "Department: " << department << endl;
        cout << "Duty: " << duty << endl;
    }
};

class Admin : public Employee {
    char role[30];
    int officeNo;

public:
    void getAdminDetails() {
        getEmployeeDetails();
        cout << "Enter Role: ";
        cin.getline(role, 30);
        cout << "Enter Office Number: ";
        cin >> officeNo;
    }

    void showAdminDetails() const {
        showEmployeeDetails();
        cout << "Role: " << role << endl;
        cout << "Office No: " << officeNo << endl;
    }
};

int main() {
    int choice;
    cout << "Select Employee Type:\n";
    cout << "1. Professor\n";
    cout << "2. Non-Teaching\n";
    cout << "3. Admin\n";
    cout << "Enter your choice: ";
    cin >> choice;

    switch (choice) {
        case 1: {
            Professor prof;
            prof.getProfessorDetails();
            cout << "\n--- Professor Details ---\n";
            prof.showProfessorDetails();
            break;
        }
        case 2: {
            NonTeaching nt;
            nt.getNonTeachingDetails();
            cout << "\n--- Non-Teaching Staff Details ---\n";
            nt.showNonTeachingDetails();
            break;
        }
        case 3: {
            Admin ad;
            ad.getAdminDetails();
            cout << "\n--- Admin Staff Details ---\n";
            ad.showAdminDetails();
            break;
        }
        default:
            cout << "Invalid choice.\n";
    }

    return 0;
}
```

OUTPUT:-

```
Select Employee Type:
1. Professor
2. Non-Teaching
3. Admin
```

```
Enter your choice: 1
Enter Employee ID: 232
Enter Name: Prerna
Enter Address: pune
Enter Subject: chemistry
Enter Qualification: phd
Enter Designation: teacher
Enter Research Area: laboratory

--- Professor Details ---
Employee ID: 232
Name: Prerna
Address: pune
Subject: chemistry
Qualification: phd
Designation: teacher
Research Area: laboratory



=== Code Execution Successful ===
```

4. Hybrid Inheritance

Consider a class network of the following figure. The class master derives information from both account and admin classes which in turn derives information from the class person. Define all the four classes and write a program to create, update and display the information contained in master objects.

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

class Person {
protected:
    char name[50];
    int code;

public:
    void getPerson() {
        cout << "Enter name: ";
        cin.ignore();
        cin.getline(name, 50);
        cout << "Enter code: ";
        cin >> code;
    }

    void showPerson() const {
        cout << "Name: " << name << endl;
        cout << "Code: " << code << endl;
    }
};
class Account : virtual public Person {
protected:
    float pay;

public:
    void getAccount() {
        cout << "Enter pay: ";
        cin >> pay;
    }

    void showAccount() const {
        cout << "Pay: " << pay << endl;
    }
};
```

```
class Admin : virtual public Person {
protected:
   int experience;

public:
   void getAdmin() {
      cout << "Enter experience (in years): ";
      cin >> experience;
   }

   void showAdmin() const {
      cout << "Experience: " << experience << " years" << endl;
   }
};
class Master : public Account, public Admin {
public:
   void create() {
      getPerson();
      getAccount();
      getAdmin();
   }

   void update() {
      cout << "\n--- Update Data ---\n";
      create(); // just re-enter all info for simplicity
   }

   void display() const {
      cout << "\n--- Displaying Master Information ---\n";
      showPerson();
```

```
        showAccount();
        showAdmin();
    }
};

int main() {
    Master m;
    int choice;

    do {
        cout << "\nMenu:\n";
        cout << "1. Create Record\n";
        cout << "2. Update Record\n";
        cout << "3. Display Record\n";
        cout << "4. Exit\n";
        cout << "Enter your choice: ";
        cin >> choice;

        switch (choice) {
            case 1:
                m.create();
                break;
            case 2:
                m.update();
                break;
            case 3:
                m.display();
                break;
            case 4:
                cout << "Exiting program.\n";
                break;
            default:
                cout << "Invalid choice.\n";
        }

    } while (choice != 4);

    return 0;
}
```

OUTPUT:-

```
Menu:
1. Create Record
2. Update Record
3. Display Record
4. Exit
Enter your choice: 1
Enter name: shetty
Enter code: 49
Enter pay: 2089
Enter experience (in years): 3

Menu:
1. Create Record
2. Update Record
3. Display Record
4. Exit
Enter your choice: 3

--- Displaying Master Information ---
Name: shetty
Code: 49
```

```
Pay: 2089
Experience: 3 years

Menu:
1. Create Record
2. Update Record
3. Display Record
4. Exit
Enter your choice: 4
Exiting program.


=== Code Execution Successful ===
```

5. BookShop

A bookshop maintains the inventory of books that are being sold at the shop. The list includes details such as author, title, price, publisher and stock position. Whenever a customer wants a book, the salesperson inputs the title and author and the system searches the list and displays whether it is available or not. If it is not, an appropriate message is displayed. If it is, then the system displays the book details and requests for the number of copies required. If the requested copies are available, the
total cost of the requested copies is displayed; otherwise “Required copies not in stock” is displayed.
Design a system using a class called books with suitable member functions and constructors. Use new
operators in constructors to allocate the memory space required.

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

class Book {
private:
    char* title;
    char* author;
    char* publisher;

    float price;

    int stock;


    int isEqual(const char* s1, const char* s2) const {

        int i = 0;

        while (s1[i] != '\0' && s2[i] != '\0') {

            if (s1[i] != s2[i])

                return 0;

            i++;

        }

        if (s1[i] == '\0' && s2[i] == '\0')

            return 1;

        else

            return 0;

    }


public:


    Book() {

        title = new char[50];

        author = new char[50];

        publisher = new char[50];

        price = 0.0;

        stock = 0;

    }




    ~Book() {

        delete[] title;

        delete[] author;
        delete[] publisher;
    }
    void input() {
        cin.ignore();
        cout << "Enter title: ";
        cin.getline(title, 50);
        cout << "Enter author: ";
        cin.getline(author, 50);
        cout << "Enter publisher: ";
        cin.getline(publisher, 50);
        cout << "Enter price: ";
        cin >> price;
        cout << "Enter stock: ";
        cin >> stock;
    }
    void display() const {
        cout << "\nTitle: " << title
             << "\nAuthor: " << author
             << "\nPublisher: " << publisher
             << "\nPrice: " << price
             << "\nStock: " << stock << endl;
    }
    int search(const char* t, const char* a) const {
        if (isEqual(title, t) && isEqual(author, a))
            return 1;
        else
            return 0;
    }
    void sell() {
        int copies;
        cout << "Enter number of copies required: ";
```

```
        cin >> copies;
        if (copies <= stock) {
            cout << "Total cost: " << copies * price << endl;
            stock -= copies;
        } else {
            cout << "Required copies not in stock.\n";
        }
    }
};

int main() {
    const int size = 3;
    Book books[size];
    int choice;
    char title[50], author[50];
    cout << "--- Enter Book Details ---\n";
    for (int i = 0; i < size; ++i) {
        cout << "\nBook " << i + 1 << ":\n";
        books[i].input();
    }
    do {
        cout << "\n--- Menu ---\n";
        cout << "1. Search and Buy a Book\n";
        cout << "2. Display All Books\n";
        cout << "3. Exit\n";
        cout << "Enter your choice: ";
        cin >> choice;

        switch (choice) {
            case 1: {
                cin.ignore();
    cout << "Enter book title: ";
    cin.getline(title, 50);
    cout << "Enter author name: ";
    cin.getline(author, 50);

    int found = 0;
    for (int i = 0; i < size; ++i) {
        if (books[i].search(title, author) == 1) {
            cout << "\nBook Found!\n";
            books[i].display();
            books[i].sell();
            found = 1;
            break;
        }
    }
    if (found == 0)
        cout << "Book not found.\n";
    break;
}

case 2:
    for (int i = 0; i < size; ++i) {
        cout << "\nBook " << i + 1 << " Details:";
        books[i].display();
    }
    break;

case 3:
    cout << "Exiting program.\n";
    break;
```

```
            default:
                cout << "Invalid choice.\n";
        }

    } while (choice != 3);

    return 0;
}
```

OUTPUT:-

```
--- Enter Book Details ---

Book 1:
Python
Enter title: Enter author: SL ARORA
Enter publisher: JK
Enter price: 400
Enter stock: 4

Book 2:
Enter title: C
Enter author: Sumita
Enter publisher: KK
Enter price: 500
Enter stock: 8

Book 3:
Enter title: Java
Enter author: Frost
Enter publisher: HH
Enter price: 980
Enter stock: 2
```

```
--- Menu ---

1. Search and Buy a Book

2. Display All Books

3. Exit

Enter your choice: 2


Book 1 Details:

Title: ython

Author: SL ARORA

Publisher: JK

Price: 400

Stock: 4


Book 2 Details:

Title: C

Author: Sumita

Publisher: KK

Price: 500

Stock: 8


Book 3 Details:

Title: Java

Author: Frost

Publisher: HH

Price: 980

Stock: 2


--- Menu ---

1. Search and Buy a Book

2. Display All Books
```

```
3. Exit
Enter your choice: 2

Book 1 Details:
Title: ython
Author: SL ARORA
Publisher: JK
Price: 400
Stock: 4

Book 2 Details:
Title: C
Author: Sumita
Publisher: KK
Price: 500
Stock: 8

Book 3 Details:
Title: Java
Author: Frost
Publisher: HH
Price: 980
Stock: 2

--- Menu ---
1. Search and Buy a Book
2. Display All Books
3. Exit
Enter your choice: 3
Exiting program.
```

```
=== Code Execution Successful ===
```

6. Create two classes DM and DB which store the value of distances. DM stores distance in meters

and centimeters and DB in feet and inches. Write a program that can read values for the class objects

and add one object of DM with another object of DB. Use a friend function to carry out the addition

operation. The object that stores the results may be a DM object or DB object, depending on the units

in which the results are required. The display should be in the format of feet and inches or meters and

centimeters depending on the object on display.

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

class DB;

class DM {
private:
   int meters;
   int centimeters;

public:
   DM() {
      meters = 0;
      centimeters = 0;
   }

   void read() {
      cout << "Enter distance in meters and centimeters:\n";
      cout << "Meters: ";
      cin >> meters;
```

```
        cout << "Centimeters: ";
        cin >> centimeters;
    }

    void display() const {
        cout << "Distance: " << meters << " meters " << centimeters << " centimeters\n";
    }

    friend DM add(const DM& d1, const DB& d2); // Friend function
};

class DB {
private:
    int feet;
    int inches;

public:
    DB() {
        feet = 0;
        inches = 0;
    }

    void read() {
        cout << "Enter distance in feet and inches:\n";
        cout << "Feet: ";
        cin >> feet;
        cout << "Inches: ";
        cin >> inches;
    }

    void display() const {
        cout << "Distance: " << feet << " feet " << inches << " inches\n";
    }

    friend DM add(const DM& d1, const DB& d2); // Same friend function
};

DM add(const DM& d1, const DB& d2) {
    float total_cm_d1 = d1.meters * 100 + d1.centimeters;
    float total_cm_d2 = d2.feet * 30.48 + d2.inches * 2.54;
    float total_cm = total_cm_d1 + total_cm_d2;

    DM result;
    result.meters = static_cast<int>(total_cm) / 100;
    result.centimeters = static_cast<int>(total_cm) % 100;

    return result;
}

int main() {
    DM d1, result_dm;
    DB d2;

    cout << "--- Input for DM Object ---\n";
    d1.read();

    cout << "\n--- Input for DB Object ---\n";
    d2.read();
    result_dm = add(d1, d2);

    cout << "\n--- Result in Meters and Centimeters ---\n";
    result_dm.display();
    return 0;
}
```

OUTPUT:-

```
--- Input for DM Object ---
Enter distance in meters and centimeters:
Meters: 200
Centimeters: 20000

--- Input for DB Object ---
Enter distance in feet and inches:
Feet: 5
Inches: 10

--- Result in Meters and Centimeters ---
Distance: 401 meters 77 centimeters



=== Code Execution Successful ===
```

7. The SimpleCircle class

1. Write a SimpleCircle class declaration (only) with one member variable: itsRadius. Include a default constructor, a destructor, and accessor methods for radius.
2. Using the class you created in Exercise 1, write the implementation of the default constructor, initializing itsRadius with the value 5.
3. Using the same class, add a second constructor that takes a value as its parameter and assigns that
value to itsRadius.
4. Create a prefix and postfix increment operator for your SimpleCircle class that increments itsRadius.

5. Provide a copy constructor for SimpleCircle.

6. Provide an assignment operator for SimpleCircle.

7. Write a program that creates two SimpleCircle objects. Use the default constructor on one and instantiate the other with the value 9. Call the increment operator on each and then print their values.

Finally, assign the second to the first and print its values.

```
→#include <iostream>

using namespace std;


class SimpleCircle {

private:

   int itsRadius;


public:

   SimpleCircle() {

      itsRadius = 5;

   }


   SimpleCircle(int radius) {

      itsRadius = radius;

   }

   ~SimpleCircle() {}


   int getRadius() {

      return itsRadius;

   }


   void setRadius(int radius) {

      itsRadius = radius;

   }
```

```
  SimpleCircle(const SimpleCircle &circle) {
    itsRadius = circle.itsRadius;
  }

  SimpleCircle& operator=(const SimpleCircle &circle) {
    if (this != &circle) {
      itsRadius = circle.itsRadius;
    }
    return *this;
  }

  SimpleCircle& operator++() {
    ++itsRadius;
    return *this;
  }

  SimpleCircle operator++(int) {
    SimpleCircle temp = *this;
    itsRadius++;
    return temp;
  }

  void display() {
    cout << "Radius: " << itsRadius << endl;
  }
};

int main() {
  int userRadius;

  SimpleCircle circle1;
```

```
    cout << "Default Circle 1 created with radius 5." << endl;

    cout << "Enter radius for Circle 2: ";
    cin >> userRadius;

    SimpleCircle circle2(userRadius);

    cout << "\nBefore increment:" << endl;
    cout << "Circle 1: "; circle1.display();
    cout << "Circle 2: "; circle2.display();

    ++circle1;      // Prefix increment
    circle2++;      // Postfix increment

    cout << "\nAfter increment:" << endl;
    cout << "Circle 1 (prefix ++): "; circle1.display();
    cout << "Circle 2 (postfix ++): "; circle2.display();

    circle1 = circle2;

    cout << "\nAfter assignment (circle1 = circle2):" << endl;
    cout << "Circle 1: "; circle1.display();
    cout << "Circle 2: "; circle2.display();

    return 0;
}
```

OUTPUT:-

```
Default Circle 1 created with radius 5.
Enter radius for Circle 2: 5

Before increment:
```

```
Circle 1: Radius: 5

Circle 2: Radius: 5


After increment:

Circle 1 (prefix ++): Radius: 6

Circle 2 (postfix ++): Radius: 6


After assignment (circle1 = circle2):

Circle 1: Radius: 6

Circle 2: Radius: 6



=== Code Execution Successful ===
```

8. Cinema

A program to deal with the day-to-day administration of bookings for a Cinema for a single day. Each

day there are three separate performances. An early afternoon performance at 1pm, an early evening

performance at 5pm, and the main performance at 8.30pm. The program should be able to handle the

booking of cinema seats for any of these three performances and supply details about the remaining

seats for a particular performance.

→
```
#include <iostream>

using namespace std;


const int TOTAL_SEATS = 50;  // Total seats per performance


class Show {

private:

    string time;

    int bookedSeats[TOTAL_SEATS];

    int seatCount;
```

```
public:
   Show(string t) {
      time = t;
      seatCount = 0;
      for (int i = 0; i < TOTAL_SEATS; i++)
         bookedSeats[i] = 0; // 0 means available
   }

   void bookSeat() {
      if (seatCount >= TOTAL_SEATS) {
         cout << "Sorry, all seats are booked for " << time << " show.\n";
         return;
      }

      int seatNo;
      cout << "Enter seat number to book (1 to " << TOTAL_SEATS << "): ";
      cin >> seatNo;

      if (seatNo < 1 || seatNo > TOTAL_SEATS) {
         cout << "Invalid seat number.\n";
         return;
      }

      if (bookedSeats[seatNo - 1] == 1) {
         cout << "Seat already booked.\n";
      } else {
         bookedSeats[seatNo - 1] = 1;
         seatCount++;
         cout << "Seat " << seatNo << " successfully booked for " << time << " show.\n";
      }
```

```
    }

    void showAvailableSeats() const {
        cout << "\nPerformance at " << time << ":\n";
        cout << "Total Seats: " << TOTAL_SEATS << "\n";
        cout << "Booked Seats: " << seatCount << "\n";
        cout << "Available Seats: " << TOTAL_SEATS - seatCount << "\n";
    }
};
int main() {
    Show matinee("1:00 PM"), evening("5:00 PM"), night("8:30 PM");
    int choice, showChoice;

    do {
        cout << "\n--- Cinema Booking Menu ---\n";
        cout << "1. Book a Seat\n";
        cout << "2. Show Available Seats\n";
        cout << "3. Exit\n";
        cout << "Enter your choice: ";
        cin >> choice;

        if (choice == 1 || choice == 2) {
            cout << "Select Show:\n";
            cout << "1. 1:00 PM\n2. 5:00 PM\n3. 8:30 PM\n";
            cout << "Enter show number: ";
            cin >> showChoice;

            Show* selectedShow = nullptr;

            if (showChoice == 1) selectedShow = &matinee;
            else if (showChoice == 2) selectedShow = &evening;
            else if (showChoice == 3) selectedShow = &night;
            else {
                cout << "Invalid show selection.\n";
                continue;
            }

            if (choice == 1)
                selectedShow->bookSeat();
            else
                selectedShow->showAvailableSeats();
        }
        else if (choice == 3) {
            cout << "Exiting booking system.\n";
        }
        else {
            cout << "Invalid choice.\n";
        }

    } while (choice != 3);

    return 0;
}
```

OUTPUT:-

```
--- Cinema Booking Menu ---
1. Book a Seat
2. Show Available Seats
3. Exit
Enter your choice: 1
Select Show:
```

```
1. 1:00 PM
2. 5:00 PM
3. 8:30 PM
Enter show number: 2
Enter seat number to book (1 to 50): 3
Seat 3 successfully booked for 5:00 PM show.

--- Cinema Booking Menu ---
1. Book a Seat
2. Show Available Seats
3. Exit
Enter your choice: 2
Select Show:
1. 1:00 PM
2. 5:00 PM
3. 8:30 PM
Enter show number: 2

Performance at 5:00 PM:
Total Seats: 50
Booked Seats: 1
Available Seats: 49

--- Cinema Booking Menu ---
1. Book a Seat
2. Show Available Seats
3. Exit
Enter your choice: 3
Exiting booking system.
```

```
=== Code Execution Successful ===
```

9. Library

A program which maintains the state of books in a small school library. Each book in the library has a class mark which is a number in the range 1 - 999. Two or more identical books will have a different class mark. A person may:

Check a book out of the library.

Reserve a book which is out on loan.

Inquire as to the current status of a book.

The program should be able to handle the above day-to-day transactions. In addition, a facility should

be included which will provide a summary about the current status of the books in the library. For example:

Books in library 100

Books on loan = 5

Books reserved = 2

Books on shelves = 93

Hint: Create a class for a book. The responsibilities of the book class are:

→

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

const int MAX_BOOKS = 100;

class Book {
private:
    int classMark;
    int status;

public:
    Book() {
        classMark = 0;
        status = 0;
    }

    void setClassMark(int cm) {
        classMark = cm;
        status = 0;
    }

    int getClassMark() const {
        return classMark;
    }

    void checkOut() {
        if (status == 0) {
            status = 1;
            cout << "Book " << classMark << " has been checked out.\n";
        } else {
            cout << "Book " << classMark << " is not available for checkout.\n";
        }
    }

    void reserve() {
        if (status == 1) {
            status = 2;
            cout << "Book " << classMark << " has been reserved.\n";
        } else if (status == 0) {
            cout << "Book is available, no need to reserve.\n";
        } else {
            cout << "Book already reserved.\n";
        }
    }
```

```
    void returnBook() {
        if (status != 0) {
            status = 0;
            cout << "Book " << classMark << " returned to shelf.\n";
        } else {
            cout << "Book is already on the shelf.\n";
        }
    }

    void showStatus() const {
        cout << "Book " << classMark << " is ";
        if (status == 0)
            cout << "on the shelf.\n";
        else if (status == 1)
            cout << "on loan.\n";
        else
            cout << "reserved.\n";
    }

    int getStatus() const {
        return status;
    }
};

int findBook(Book books[], int total, int classMark) {
    for (int i = 0; i < total; ++i) {
        if (books[i].getClassMark() == classMark)
            return i;
    }
    return -1;
```

```
}

int main() {
    Book books[MAX_BOOKS];
    int totalBooks = 0;
    int choice, classMark;
    cout << "Enter number of books to add (max 100): ";
    cin >> totalBooks;

    for (int i = 0; i < totalBooks; ++i) {
        cout << "Enter class mark for book " << i + 1 << " (1-999): ";
        cin >> classMark;
        books[i].setClassMark(classMark);
    }

    do {
        cout << "\n--- Library Menu ---\n";
        cout << "1. Check Out Book\n";
        cout << "2. Reserve Book\n";
        cout << "3. Return Book\n";
        cout << "4. Inquire Status\n";
        cout << "5. Library Summary\n";
        cout << "6. Exit\n";
        cout << "Enter choice: ";
        cin >> choice;

        if (choice >= 1 && choice <= 4) {
            cout << "Enter class mark of book: ";
            cin >> classMark;
            int index = findBook(books, totalBooks, classMark);
            if (index == -1) {
                cout << "Book not found.\n";
                continue;
            }

            if (choice == 1)
                books[index].checkOut();
            else if (choice == 2)
                books[index].reserve();
            else if (choice == 3)
                books[index].returnBook();
            else if (choice == 4)
                books[index].showStatus();
        }

        else if (choice == 5) {
            int onLoan = 0, reserved = 0, available = 0;
            for (int i = 0; i < totalBooks; ++i) {
                int stat = books[i].getStatus();
                if (stat == 0)
                    available++;
                else if (stat == 1)
                    onLoan++;
                else if (stat == 2)
                    reserved++;
            }

            cout << "\n--- Library Summary ---\n";
            cout << "Total books: " << totalBooks << endl;
            cout << "Books on loan: " << onLoan << endl;
            cout << "Books reserved: " << reserved << endl;
```

```
            cout << "Books on shelves: " << available << endl;
        }

        else if (choice == 6) {
            cout << "Exiting library system.\n";
        }

        else {
            cout << "Invalid choice.\n";
        }

    } while (choice != 6);

    return 0;
}
```

OUTPUT:-

```
Enter number of books to add (max 100): 10
Enter class mark for book 1 (1-999): 99
Enter class mark for book 2 (1-999): 32
Enter class mark for book 3 (1-999): 44
Enter class mark for book 4 (1-999): 34
Enter class mark for book 5 (1-999): 56
Enter class mark for book 6 (1-999): 65
Enter class mark for book 7 (1-999): 43
Enter class mark for book 8 (1-999): 1
Enter class mark for book 9 (1-999): 3
Enter class mark for book 10 (1-999): 45

--- Library Menu ---
1. Check Out Book
```

```
2. Reserve Book
3. Return Book
4. Inquire Status
5. Library Summary
6. Exit
Enter choice: 1
Enter class mark of book: 32
Book 32 has been checked out.

--- Library Menu ---
1. Check Out Book
2. Reserve Book
3. Return Book
4. Inquire Status
5. Library Summary
6. Exit
Enter choice: 6
Exiting library system.


=== Code Execution Successful ===
```

10. The Employee class

1. Write the code that declares a class called Employee with these data members: age, yearsOfService,

and Salary.

2. Rewrite the Employee class to make the data members private, and provide public accessor methods

to get and set each of the data members.

3. Write a program with the Employee class that makes two Employees; sets their age,

YearsOfService, and Salary; and prints their values.

4. Continuing from Exercise 3, provide a method of Employee that reports how many thousands of

dollars the employee earns, rounded to the nearest 1,000.

5. Change the Employee class so that you can initialize age, YearsOfService, and Salary when you create the employee.

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

class Employee {
private:
   int age;
   int yearsOfService;
   double salary;

public:
   Employee(int a, int y, double s) {
      age = a;
      yearsOfService = y;
      salary = s;
   }

   Employee() {
      age = 0;
      yearsOfService = 0;
      salary = 0.0;
   }

   void setAge(int a) {
      age = a;
   }

   void setYearsOfService(int y) {
      yearsOfService = y;
```

```
    }

    void setSalary(double s) {
        salary = s;
    }

    int getAge() const {
        return age;
    }

    int getYearsOfService() const {
        return yearsOfService;
    }

    double getSalary() const {
        return salary;
    }

    int getSalaryInThousands() const {
        return static_cast<int>((salary + 500) / 1000); // round to nearest 1000
    }

    void printDetails() const {
        cout << "Age: " << age << "\n";
        cout << "Years of Service: " << yearsOfService << "\n";
        cout << "Salary: $" << salary << "\n";
        cout << "Rounded Salary (Thousands): $" << getSalaryInThousands() << "000\n";
        cout << "----------------------\n";
    }
};
```

```
int main() {
    Employee emp1, emp2;
    int age, service;
    double salary;

    cout << "Enter details for Employee 1:\n";
    cout << "Age: ";
    cin >> age;
    cout << "Years of Service: ";
    cin >> service;
    cout << "Salary: ";
    cin >> salary;
    emp1 = Employee(age, service, salary);

    cout << "\nEnter details for Employee 2:\n";
    cout << "Age: ";
    cin >> age;
    cout << "Years of Service: ";
    cin >> service;
    cout << "Salary: ";
    cin >> salary;
    emp2 = Employee(age, service, salary);

    cout << "\n--- Employee 1 ---\n";
    emp1.printDetails();

    cout << "--- Employee 2 ---\n";
    emp2.printDetails();

    return 0;
}
```

OUTPUT:-

```
Enter details for Employee 1:

Age: 25

Years of Service: 3

Salary: 100000


Enter details for Employee 2:

Age: 23

Years of Service: 2

Salary: 60000


--- Employee 1 ---

Age: 25

Years of Service: 3

Salary: $100000

Rounded Salary (Thousands): $100000

----------------------

--- Employee 2 ---

Age: 23

Years of Service: 2

Salary: $60000

Rounded Salary (Thousands): $60000

----------------------




=== Code Execution Successful ===
```

11.Create two classes named Mammals and MarineAnimals. Create another class named BlueWhale which inherits both the above classes. Now, create a function in each of these classes which prints "I am mammal", "I am a marine animal" and "I belong to both the categories: Mammals as well as

Marine Animals" respectively. Now, create an object for each of the above class and try calling

1 - function of Mammals by the object of Mammal

2 - function of MarineAnimal by the object of MarineAnimal

3 - function of BlueWhale by the object of BlueWhale

4 - function of each of its parent by the object of BlueWhale

→

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

class Mammals {
public:
   void display() {
      cout << "I am mammal" << endl;
   }
};

class MarineAnimals {
public:
   void display() {
      cout << "I am a marine animal" << endl;
   }
};

class BlueWhale : public Mammals, public MarineAnimals {
public:
   void display() {
      cout << "I belong to both the categories: Mammals as well as Marine Animals" << endl;
   }
};

int main() {
   Mammals m;
```

```
    MarineAnimals ma;
    BlueWhale bw;
    cout << "Mammals object says: ";
    m.display();



    cout << "MarineAnimals object says: ";
    ma.display();

    cout << "BlueWhale object says: ";
    bw.display();
    cout << "Calling Mammals function from BlueWhale object: ";
    bw.Mammals::display();

    cout << "Calling MarineAnimals function from BlueWhale object: ";
    bw.MarineAnimals::display();

    return 0;
}
```

```
OUTPUT:-Mammals object says: I am mammal
MarineAnimals object says: I am a marine animal
BlueWhale object says: I belong to both the categories: Mammals as well as Marine Animals
Calling Mammals function from BlueWhale object: I am mammal
Calling MarineAnimals function from BlueWhale object: I am a marine animal
```

12. Make a class named Fruit with a data member to calculate the number of fruits in a basket. Create

two other class named Apples and Mangoes to calculate the number of apples and mangoes in the

basket. Print the number of fruits of each type and the total number of fruits in the basket.

```
→#include <iostream>
```

```
using namespace std;

class Fruit {
protected:
   int totalFruits;

public:
   Fruit() {
      totalFruits = 0;
   }

   void addToTotal(int count) {
      totalFruits += count;
   }

   void displayTotalFruits() {
      cout << "Total fruits in basket: " << totalFruits << endl;
   }
};

class Apples : public Fruit {
private:
   int numApples;

public:
   Apples(int n) {
      numApples = n;
      addToTotal(n);  // update base class total
   }

   void displayApples() {
        cout << "Number of apples: " << numApples << endl;
    }

    int getAppleCount() {
        return numApples;
    }
};

class Mangoes : public Fruit {
private:
    int numMangoes;

public:
    Mangoes(int n) {
        numMangoes = n;
        addToTotal(n);  // update base class total
    }

    void displayMangoes() {
        cout << "Number of mangoes: " << numMangoes << endl;
    }

    int getMangoCount() {
        return numMangoes;
    }
};

int main() {
    int appleCount, mangoCount;

    cout << "Enter number of apples: ";
    cin >> appleCount;

    cout << "Enter number of mangoes: ";
    cin >> mangoCount;

    Apples a(appleCount);
    Mangoes m(mangoCount);

    cout << "\n--- Fruit Basket Summary ---\n";
    a.displayApples();
    m.displayMangoes();
    cout << "Total fruits in basket: " << (a.getAppleCount() + m.getMangoCount()) << endl;

    return 0;
}
```

OUTPUT:-

```
Enter number of apples: 5
Enter number of mangoes: 4

--- Fruit Basket Summary ---
Number of apples: 5
Number of mangoes: 4
Total fruits in basket: 9


=== Code Execution Successful ===
```

13. We want to calculate the total marks of each student of a class in Physics,Chemistry and

Mathematics and the average marks of the class. The number of students in the class are entered by the user. Create a class named Marks with data members for roll number, name and marks. Create three other classes inheriting the Marks class, namely Physics, Chemistry and Mathematics, which are used to define marks in individual subject of each student. Roll number of each student will be generated automatically.

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

int rollCounter = 1;  // Global counter for roll number generation


class Marks {
protected:
   int rollNo;
   char name[50];


public:
   Marks() {
      rollNo = rollCounter++;
   }


   void inputName() {
      cout << "Enter name: ";
      cin >> name;
   }


   void displayName() {
      cout << "Roll No: " << rollNo << ", Name: " << name;
   }


   int getRollNo() {
      return rollNo;
   }
```

```
  const char* getName() {
    return name;
  }
};

class Physics : public Marks {
protected:
  int physicsMarks;

public:
  void inputPhysicsMarks() {
    inputName();
    cout << "Enter Physics marks: ";
    cin >> physicsMarks;
  }

  int getPhysicsMarks() {
    return physicsMarks;
  }
};

class Chemistry : public Physics {
protected:
  int chemistryMarks;

public:
  void inputChemistryMarks() {
    inputPhysicsMarks();
    cout << "Enter Chemistry marks: ";
    cin >> chemistryMarks;
    }

    int getChemistryMarks() {
        return chemistryMarks;
    }
};

class Mathematics : public Chemistry {
protected:
    int mathMarks;

public:
    void inputAllMarks() {
        inputChemistryMarks();
        cout << "Enter Mathematics marks: ";
        cin >> mathMarks;
    }

    int getMathMarks() {
        return mathMarks;
    }

    int getTotalMarks() {
        return getPhysicsMarks() + getChemistryMarks() + getMathMarks();
    }

    void displayStudentSummary() {
        displayName();
        cout << ", Physics: " << getPhysicsMarks()
             << ", Chemistry: " << getChemistryMarks()
             << ", Mathematics: " << getMathMarks()
```

```
            << ", Total: " << getTotalMarks() << endl;
    }
};

int main() {
    int num;
    cout << "Enter number of students: ";
    cin >> num;

    Mathematics students[num];
    int totalPhysics = 0, totalChem = 0, totalMath = 0;

    for (int i = 0; i < num; i++) {
        cout << "\nEntering details for Student " << i + 1 << ":\n";
        students[i].inputAllMarks();
        totalPhysics += students[i].getPhysicsMarks();
        totalChem += students[i].getChemistryMarks();
        totalMath += students[i].getMathMarks();
    }

    cout << "\n---- Students Summary ----\n";
    for (int i = 0; i < num; i++) {
        students[i].displayStudentSummary();
    }

    double avgPhysics = (double)totalPhysics / num;
    double avgChem = (double)totalChem / num;
    double avgMath = (double)totalMath / num;
    double avgTotal = (avgPhysics + avgChem + avgMath);

    cout << "\n--- Class Averages ---\n";
    cout << "Physics Average: " << avgPhysics << endl;
    cout << "Chemistry Average: " << avgChem << endl;
    cout << "Mathematics Average: " << avgMath << endl;
    cout << "Overall Average (Total Marks / Student): " << avgTotal << endl;

    return 0;
}
```

OUTPUT:-

```
Enter number of students: 2

Entering details for Student 1:
Enter name: wini
Enter Physics marks: 87
Enter Chemistry marks: 92
Enter Mathematics marks: 99

Entering details for Student 2:
Enter name: atharva
Enter Physics marks: 80
Enter Chemistry marks: 88
Enter Mathematics marks: 97

---- Students Summary ----
Roll No: 1, Name: wini, Physics: 87, Chemistry: 92, Mathematics: 99, Total: 278
Roll No: 2, Name: atharva, Physics: 80, Chemistry: 88, Mathematics: 97, Total: 265

--- Class Averages ---
Physics Average: 83.5
Chemistry Average: 90
Mathematics Average: 98
```

```
Overall Average (Total Marks / Student): 271.5



=== Code Execution Successful ===
```

14. We want to store the information of different vehicles. Create a class named Vehicle with two data

member named mileage and price. Create its two subclasses

*Car with data members to store ownership cost, warranty (by years), seating capacity and fuel type

(diesel or petrol).

*Bike with data members to store the number of cylinders, number of gears, cooling type(air, liquid or

oil), wheel type(alloys or spokes) and fuel tank size(in inches)

Make another two subclasses Audi and Ford of Car, each having a data member to store the model

type. Next, make two subclasses Bajaj and TVS, each having a data member to store the make-type.

Now, store and print the information of an Audi and a Ford car (i.e. model type, ownership cost,

warranty, seating capacity, fuel type, mileage and price.) Do the same for a Bajaj and a TVS bike.

→

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

class Vehicle {
protected:
   float mileage;
   float price;

public:
   void inputVehicle() {
      cout << "Enter mileage: ";
      cin >> mileage;
      cout << "Enter price: ";
      cin >> price;
   }
```

```
    void displayVehicle() {
        cout << "Mileage: " << mileage << " km/l\n";
        cout << "Price: $" << price << endl;
    }
};

class Car : public Vehicle {
protected:
    float ownershipCost;
    int warrantyYears;
    int seatingCapacity;
    char fuelType[10]; // diesel or petrol

public:
    void inputCar() {
        inputVehicle();
        cout << "Enter ownership cost: ";
        cin >> ownershipCost;
        cout << "Enter warranty (in years): ";
        cin >> warrantyYears;
        cout << "Enter seating capacity: ";
        cin >> seatingCapacity;
        cout << "Enter fuel type (diesel/petrol): ";
        cin >> fuelType;
    }

    void displayCar() {
        displayVehicle();
        cout << "Ownership Cost: $" << ownershipCost << endl;
        cout << "Warranty: " << warrantyYears << " years\n";
        cout << "Seating Capacity: " << seatingCapacity << endl;

        cout << "Fuel Type: " << fuelType << endl;

    }

};


class Audi : public Car {

    char modelType[20];


public:

    void inputAudi() {

        cout << "\n--- Enter Audi Car Info ---\n";

        inputCar();

        cout << "Enter model type: ";

        cin >> modelType;

    }


    void displayAudi() {

        cout << "\n--- Audi Car Info ---\n";

        cout << "Model Type: " << modelType << endl;

        displayCar();

    }

};


class Ford : public Car {

    char modelType[20];


public:

    void inputFord() {

        cout << "\n--- Enter Ford Car Info ---\n";

        inputCar();

        cout << "Enter model type: ";
        cin >> modelType;
    }

    void displayFord() {
        cout << "\n--- Ford Car Info ---\n";
        cout << "Model Type: " << modelType << endl;
        displayCar();
    }
};


class Bike : public Vehicle {
protected:
    int numCylinders;
    int numGears;
    char coolingType[10];
    char wheelType[10];
    float fuelTankSize;

public:
    void inputBike() {
        inputVehicle();
        cout << "Enter number of cylinders: ";
        cin >> numCylinders;
        cout << "Enter number of gears: ";
        cin >> numGears;
        cout << "Enter cooling type (air/liquid/oil): ";
        cin >> coolingType;
        cout << "Enter wheel type (alloys/spokes): ";
        cin >> wheelType;
        cout << "Enter fuel tank size (in inches): ";
        cin >> fuelTankSize;
    }

    void displayBike() {
        displayVehicle();
        cout << "Cylinders: " << numCylinders << ", Gears: " << numGears << endl;
        cout << "Cooling Type: " << coolingType << ", Wheel Type: " << wheelType << endl;
        cout << "Fuel Tank Size: " << fuelTankSize << " inches\n";
    }
};

class Bajaj : public Bike {
    char makeType[20];

public:
    void inputBajaj() {
        cout << "\n--- Enter Bajaj Bike Info ---\n";
        inputBike();
        cout << "Enter make type: ";
        cin >> makeType;
    }

    void displayBajaj() {
        cout << "\n--- Bajaj Bike Info ---\n";
        cout << "Make Type: " << makeType << endl;
        displayBike();
    }
};

class TVS : public Bike {
    char makeType[20];
```

```
public:
   void inputTVS() {
      cout << "\n--- Enter TVS Bike Info ---\n";
      inputBike();
      cout << "Enter make type: ";
      cin >> makeType;
   }

   void displayTVS() {
      cout << "\n--- TVS Bike Info ---\n";
      cout << "Make Type: " << makeType << endl;
      displayBike();
   }
};

int main() {
   Audi a;
   Ford f;
   Bajaj b;
   TVS t;

   a.inputAudi();
   f.inputFord();
   b.inputBajaj();
   t.inputTVS();

   a.displayAudi();
   f.displayFord();
   b.displayBajaj();
   t.displayTVS();
```

```
    return 0;
}
```

OUTPUT:-

```
--- Enter Audi Car Info ---
Enter mileage: 50
Enter price: 100000
Enter ownership cost: 90000
Enter warranty (in years): 3
Enter seating capacity: 3
Enter fuel type (diesel/petrol): petrol
Enter model type: a1

--- Enter Ford Car Info ---
Enter mileage: 45
Enter price: 250000
Enter ownership cost: 300000
Enter warranty (in years): 4
Enter seating capacity: 4
Enter fuel type (diesel/petrol): diesel
Enter model type: f4

--- Enter Bajaj Bike Info ---
Enter mileage: 35
Enter price: 30000
Enter number of cylinders: 2
Enter number of gears: 4
Enter cooling type (air/liquid/oil): air
Enter wheel type (alloys/spokes): alloys
Enter fuel tank size (in inches): 4
Enter make type: d9

--- Enter TVS Bike Info ---
Enter mileage: 30
Enter price: 245600
Enter number of cylinders: 4
Enter number of gears: 6
Enter cooling type (air/liquid/oil): oil
Enter wheel type (alloys/spokes): spokes
Enter fuel tank size (in inches): 3
Enter make type: f3

--- Audi Car Info ---
Model Type: a1
Mileage: 50 km/l
Price: $100000
Ownership Cost: $90000
Warranty: 3 years
Seating Capacity: 3
Fuel Type: petrol

--- Ford Car Info ---
Model Type: f4
Mileage: 45 km/l
Price: $250000
Ownership Cost: $300000
Warranty: 4 years
Seating Capacity: 4
Fuel Type: diesel

--- Bajaj Bike Info ---
```

```
Make Type: d9
Mileage: 35 km/l
Price: $30000
Cylinders: 2, Gears: 4
Cooling Type: air, Wheel Type: alloys
Fuel Tank Size: 4 inches

--- TVS Bike Info ---
Make Type: f3
Mileage: 30 km/l
Price: $245600
Cylinders: 4, Gears: 6
Cooling Type: oil, Wheel Type: spokes
Fuel Tank Size: 3 inches



=== Code Execution Successful ===
```

15. Create a class named Shape with a function that prints "This is a shape". Create another class named Polygon inheriting the Shape class with the same function that prints "Polygon is a shape". Create two other classes named Rectangle and Triangle having the same function which prints "Rectangle is a polygon" and "Triangle is a polygon" respectively. Again, make another class named Square having the same function which prints "Square is a rectangle".
Now, try calling the function by the object of each of these classes.

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

class Shape {
public:
   void display() {
       cout << "This is a shape" << endl;
```

```
    }
};

class Polygon : public Shape {
public:
    void display() {
        cout << "Polygon is a shape" << endl;
    }
};

class Rectangle : public Polygon {
public:
    void display() {
        cout << "Rectangle is a polygon" << endl;
    }
};

class Triangle : public Polygon {
public:
    void display() {
        cout << "Triangle is a polygon" << endl;
    }
};

class Square : public Rectangle {
public:
    void display() {
        cout << "Square is a rectangle" << endl;
    }
};
```

```
int main() {
   Shape s;
   Polygon p;
   Rectangle r;
   Triangle t;
   Square sq;

   cout << "\nCalling display() for each object:\n";
   s.display();
   p.display();
   r.display();
   t.display();
   sq.display();

   return 0;
}
```

OUTPUT:-

```
Calling display() for each object:
This is a shape
Polygon is a shape
Rectangle is a polygon
Triangle is a polygon
Square is a rectangle
```

16. All the banks operating in India are controlled by RBI. RBI has set a well defined guideline (e.g. minimum interest rate, minimum balance allowed, maximum withdrawal limit etc) which all banks must follow. For example, suppose RBI has set minimum interest rate applicable to a saving bank account to be 4% annually; however, banks are free to use 4% interest rate or to set any rates above it.

→
```
#include <iostream>
```

```
using namespace std;
class RBI {
protected:
   float minInterestRate;
   float minBalance;
   float maxWithdrawalLimit;

public:
   RBI() {
       minInterestRate = 4.0;      // Minimum interest rate in %
       minBalance = 1000.0;        // Minimum balance in ₹
       maxWithdrawalLimit = 25000;  // Maximum withdrawal in ₹
   }

   void showRBIRules() {
       cout << "\n--- RBI Guidelines ---\n";
       cout << "Minimum Interest Rate: " << minInterestRate << "%\n";
       cout << "Minimum Balance: ₹" << minBalance << endl;
       cout << "Maximum Withdrawal Limit: ₹" << maxWithdrawalLimit << endl;
   }

   float getMinInterest() { return minInterestRate; }
   float getMinBalance() { return minBalance; }
   float getMaxWithdrawal() { return maxWithdrawalLimit; }
};
class SBI : public RBI {
   float interestRate;
   float minBalanceBank;
   float withdrawalLimit;

public:
```

```
void inputPolicies() {
    cout << "\nEnter SBI Bank Policies:\n";
    cout << "Enter interest rate (%): ";
    cin >> interestRate;
    cout << "Enter minimum balance (₹): ";
    cin >> minBalanceBank;
    cout << "Enter maximum withdrawal limit (₹): ";
    cin >> withdrawalLimit;
}

void showPolicies() {
    cout << "\n--- SBI Bank Policies ---\n";
    cout << "Interest Rate: " << interestRate << "%\n";
    cout << "Minimum Balance: ₹" << minBalanceBank << endl;
    cout << "Withdrawal Limit: ₹" << withdrawalLimit << endl;

    cout << "\nPolicy Compliance Check:\n";
    if (interestRate >= getMinInterest())
        cout << "✔ Interest rate follows RBI guidelines.\n";
    else
        cout << "✗ Interest rate is below RBI minimum.\n";

    if (minBalanceBank >= getMinBalance())
        cout << "✔ Minimum balance follows RBI guidelines.\n";
    else
        cout << "✗ Minimum balance is below RBI requirement.\n";

    if (withdrawalLimit <= getMaxWithdrawal())
        cout << "✔ Withdrawal limit follows RBI guidelines.\n";
    else
        cout << "✗ Withdrawal limit exceeds RBI maximum.\n";
    }
};

class ICICI : public RBI {
    float interestRate;
    float minBalanceBank;
    float withdrawalLimit;

public:
    void inputPolicies() {
        cout << "\nEnter ICICI Bank Policies:\n";
        cout << "Enter interest rate (%): ";
        cin >> interestRate;
        cout << "Enter minimum balance (₹): ";
        cin >> minBalanceBank;
        cout << "Enter maximum withdrawal limit (₹): ";
        cin >> withdrawalLimit;
    }

    void showPolicies() {
        cout << "\n--- ICICI Bank Policies ---\n";
        cout << "Interest Rate: " << interestRate << "%\n";
        cout << "Minimum Balance: ₹" << minBalanceBank << endl;
        cout << "Withdrawal Limit: ₹" << withdrawalLimit << endl;

        cout << "\nPolicy Compliance Check:\n";
        if (interestRate >= getMinInterest())
            cout << "✓ Interest rate follows RBI guidelines.\n";
        else
            cout << "✗ Interest rate is below RBI minimum.\n";
        if (minBalanceBank >= getMinBalance())
            cout << "✓ Minimum balance follows RBI guidelines.\n";
        else
            cout << "✗ Minimum balance is below RBI requirement.\n";

        if (withdrawalLimit <= getMaxWithdrawal())
            cout << "✓ Withdrawal limit follows RBI guidelines.\n";
        else
            cout << "✗ Withdrawal limit exceeds RBI maximum.\n";
    }
};

int main() {
    RBI rbi;
    SBI sbi;
    ICICI icici;

    rbi.showRBIRules();

    sbi.inputPolicies();
    icici.inputPolicies();

    sbi.showPolicies();
    icici.showPolicies();

    return 0;
}
```

OUTPUT:-

```
--- RBI Guidelines ---
Minimum Interest Rate: 4%
Minimum Balance: ₹1000
Maximum Withdrawal Limit: ₹25000

Enter SBI Bank Policies:
Enter interest rate (%): 4
Enter minimum balance (₹): 10000
Enter maximum withdrawal limit (₹): 2000

Enter ICICI Bank Policies:
Enter interest rate (%): 3
Enter minimum balance (₹): 49493
Enter maximum withdrawal limit (₹): 3333

--- SBI Bank Policies ---
Interest Rate: 4%
Minimum Balance: ₹10000
Withdrawal Limit: ₹2000

Policy Compliance Check:
✓ Interest rate follows RBI guidelines.
✓ Minimum balance follows RBI guidelines.
✓ Withdrawal limit follows RBI guidelines.

--- ICICI Bank Policies ---
Interest Rate: 3%
Minimum Balance: ₹49493
Withdrawal Limit: ₹3333

Policy Compliance Check:
✗ Interest rate is below RBI minimum.
✓ Minimum balance follows RBI guidelines.
✓ Withdrawal limit follows RBI guidelines.



=== Code Execution Successful ===
```

17. Write a program to implement bank functionality in the above scenario. Note: Create few classes namely Customer, Account, RBI (Base Class) and few derived classes (SBI, ICICI, PNB etc). Assume and implement required member variables and functions in each class.

→
```
#include <iostream>
using namespace std;
class RBI {
protected:
   float minInterestRate;
   float minBalance;
   float maxWithdrawal;

public:
   RBI() {
      minInterestRate = 4.0;
      minBalance = 1000.0;
      maxWithdrawal = 25000.0;
   }

   void showRBIRules() {
      cout << "\n--- RBI Guidelines ---\n";
      cout << "Minimum Interest Rate: " << minInterestRate << "%\n";
      cout << "Minimum Balance: ₹" << minBalance << endl;
      cout << "Maximum Withdrawal Limit: ₹" << maxWithdrawal << endl;
   }

   float getMinInterestRate() { return minInterestRate; }
```

```
    float getMinBalance() { return minBalance; }

    float getMaxWithdrawal() { return maxWithdrawal; }

};


class Customer {

protected:

    string name;

    int age;

    int customerID;


public:

    void inputCustomerDetails() {

        cout << "Enter Customer Name: ";

        cin >> name;

        cout << "Enter Age: ";

        cin >> age;

        cout << "Enter Customer ID: ";

        cin >> customerID;

    }


    void showCustomerDetails() {

        cout << "\n--- Customer Details ---\n";

        cout << "Name: " << name << "\nAge: " << age << "\nCustomer ID: " << customerID << endl;

    }

};

class Account : public Customer {

protected:

    float balance;

    float interestRate;


public:
```

```
void createAccount(float irate) {
   interestRate = irate;
   cout << "Enter initial deposit amount: ₹";
   cin >> balance;
}

void deposit() {
   float amt;
   cout << "Enter deposit amount: ₹";
   cin >> amt;
   balance += amt;
   cout << "New Balance: ₹" << balance << endl;
}

void withdraw(float maxLimit) {
   float amt;
   cout << "Enter withdrawal amount: ₹";
   cin >> amt;
   if (amt <= balance && amt <= maxLimit) {
      balance -= amt;
      cout << "Withdrawal successful. Remaining Balance: ₹" << balance << endl;
   } else {
      cout << "Withdrawal failed! Amount exceeds limit or insufficient balance.\n";
   }
}

void showAccount() {
   cout << "\n--- Account Info ---\n";
   cout << "Balance: ₹" << balance << "\nInterest Rate: " << interestRate << "%\n";
}
```

```
    float getBalance() { return balance; }
    float getInterestRate() { return interestRate; }
};

class SBI : public RBI, public Account {
public:
    void openSBIAccount() {
        cout << "\nOpening SBI Account\n";
        inputCustomerDetails();
        createAccount(4.5); // SBI gives 4.5%
    }

    void checkCompliance() {
        cout << "\n--- SBI Policy Compliance ---\n";
        if (interestRate >= getMinInterestRate())
            cout << "✓ Interest rate OK\n";
        else
            cout << "✗ Interest rate below RBI minimum\n";

        if (balance >= getMinBalance())
            cout << "✓ Minimum balance OK\n";
        else
            cout << "✗ Minimum balance not maintained\n";
    }
};

class ICICI : public RBI, public Account {
public:
    void openICICIAccount() {
        cout << "\nOpening ICICI Account\n";
        inputCustomerDetails();
        createAccount(5.0); // ICICI gives 5.0%
    }

    void checkCompliance() {
        cout << "\n--- ICICI Policy Compliance ---\n";
        if (interestRate >= getMinInterestRate())
            cout << "✓ Interest rate OK\n";
        else
            cout << "✗ Interest rate below RBI minimum\n";

        if (balance >= getMinBalance())
            cout << "✓ Minimum balance OK\n";
        else
            cout << "✗ Minimum balance not maintained\n";
    }
};
class PNB : public RBI, public Account {
public:
    void openPNBAccount() {
        cout << "\nOpening PNB Account\n";
        inputCustomerDetails();
        createAccount(4.25); // PNB gives 4.25%
    }

    void checkCompliance() {
        cout << "\n--- PNB Policy Compliance ---\n";
        if (interestRate >= getMinInterestRate())
            cout << "✓ Interest rate OK\n";
        else
            cout << "✗ Interest rate below RBI minimum\n";
        if (balance >= getMinBalance())
            cout << "✓ Minimum balance OK\n";
        else
            cout << "✗ Minimum balance not maintained\n";
    }
};

int main() {
    int choice;
    SBI sbi;
    ICICI icici;
    PNB pnb;

    cout << "----- Welcome to Bank System -----\n";
    cout << "1. Create SBI Account\n";
    cout << "2. Create ICICI Account\n";
    cout << "3. Create PNB Account\n";
    cout << "Enter your choice: ";
    cin >> choice;

    switch (choice) {
        case 1:
            sbi.openSBIAccount();
            sbi.showCustomerDetails();
            sbi.showAccount();
            sbi.deposit();
            sbi.withdraw(sbi.getMaxWithdrawal());
            sbi.checkCompliance();
            break;

        case 2:
            icici.openICICIAccount();
            icici.showCustomerDetails();
            icici.showAccount();
            icici.deposit();
            icici.withdraw(icici.getMaxWithdrawal());
            icici.checkCompliance();
            break;

        case 3:
            pnb.openPNBAccount();
            pnb.showCustomerDetails();
            pnb.showAccount();
            pnb.deposit();
            pnb.withdraw(pnb.getMaxWithdrawal());
            pnb.checkCompliance();
            break;

        default:
            cout << "Invalid choice!" << endl;
    }

    return 0;
}
```

OUTPUT:-

```
----- Welcome to Bank System -----
1. Create SBI Account
2. Create ICICI Account
3. Create PNB Account
Enter your choice: 1
```

```
Opening SBI Account
Enter Customer Name: dishan
Enter Age: 37
Enter Customer ID: 007
Enter initial deposit amount: ₹2000

--- Customer Details ---
Name: dishan
Age: 37
Customer ID: 7

--- Account Info ---
Balance: ₹2000
Interest Rate: 4.5%
Enter deposit amount: ₹200
New Balance: ₹2200
Enter withdrawal amount: ₹100
Withdrawal successful. Remaining Balance: ₹2100

--- SBI Policy Compliance ---
✓ Interest rate OK
✓ Minimum balance OK



=== Code Execution Successful ===
```

18. Write a program to print the names of students by creating a Student class. If no name is passed

while creating an object of the Student class, then the name should be "Unknown", otherwise the name

should be equal to the String value passed while creating the object of the Student class.

```
→#include <iostream>
```

```
using namespace std;

class Student {
private:
    char name[100];

public:
    Student(const char* studentName = "Unknown") {

        int i = 0;
        while (studentName[i] != '\0' && i < 99) {
            name[i] = studentName[i];
            i++;
        }
        name[i] = '\0';
    }

        void display() {
        cout << "Student Name: " << name << endl;
    }
};

int main() {
    Student student1("John Doe");
    Student student2;



    student1.display();
    student2.display();

    return 0;
}
```

OUTPUT:-

```
Student Name: John Doe

Student Name: Unknown



=== Code Execution Successful ===
```

19. Create a class named 'Rectangle' with two data members- length and breadth and a function to calculate the area which is 'length*breadth'. The class has three constructors which are :

1 - having no parameter - values of both length and breadth are assigned zero.

2 - having two numbers as parameters - the two numbers are assigned as length and breadth respectively.

3 - having one number as parameter - both length and breadth are assigned that number.

Now, create objects of the 'Rectangle' class having none, one and two parameters and print their areas.

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

class Rectangle {
private:
   float length, breadth;

public:
   Rectangle() {
      length = 0;
      breadth = 0;
   }

   Rectangle(float l, float b) {
        length = l;
        breadth = b;
    }
    Rectangle(float side) {
        length = side;
        breadth = side;
    }

    float calculateArea() {
        return length * breadth;
    }

    void inputDimensions() {
        cout << "Enter length: ";
        cin >> length;
        cout << "Enter breadth: ";
        cin >> breadth;
    }
};

int main() {
    float l, b, side;

    Rectangle rect1;



    Rectangle rect2;
    rect2.inputDimensions();
    cout << "Enter side for square: ";
    cin >> side;
    Rectangle rect3(side);
    cout << "Area of rect1 (default constructor): " << rect1.calculateArea() << endl;
    cout << "Area of rect2 (user input): " << rect2.calculateArea() << endl;
    cout << "Area of rect3 (square with side " << side << "): " << rect3.calculateArea() << endl;

    return 0;
}
```

OUTPUT:-

```
Enter length: 5
Enter breadth: 4
Enter side for square: 2
Area of rect1 (default constructor): 0
Area of rect2 (user input): 20
Area of rect3 (square with side 2): 4



=== Code Execution Successful ===
```

20. Suppose you have a Piggie Bank with an initial amount of $50 and you have to add some more amount to it. Create a class 'AddAmount' with a data member named 'amount' with an initial value of

$50. Now make two constructors of this class as follows:

1 - without any parameter - no amount will be added to the Piggie Bank

2 - having a parameter which is the amount that will be added to the Piggie Bank

Create an object of the 'AddAmount' class and display the final amount in the Piggie Bank.

→

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

class AddAmount {
private:
    float amount;

public:
    AddAmount() {
        amount = 50.0;  // Initial amount is $50
    }
    AddAmount(float addedAmount) {
        amount = 50.0 + addedAmount;  // Initial amount + added amount
    }
    void displayAmount() {
        cout << "Final amount in the Piggie Bank: $" << amount << endl;
    }
};

int main() {
    float addedAmount;
    AddAmount bank1;
    bank1.displayAmount();  // Display initial amount of $50
    cout << "Enter the amount to add to the Piggie Bank: $";
    cin >> addedAmount;



    AddAmount bank2(addedAmount);
    bank2.displayAmount();
    return 0;
}
```

OUTPUT:-

```
Final amount in the Piggie Bank: $50
Enter the amount to add to the Piggie Bank: $100
Final amount in the Piggie Bank: $150
```

```
=== Code Execution Successful ===
```