Assignment 1

Design a mini ATM interface that simulates balance inquiry,deposit,and withdrawal using encapsulation and basic C++ constructs.

Program:

```
#include<iostream.h>

#include<conio.h>

class ATM

{

        public:

                int bal;

                ATM(float initialbal)

                {

                        bal=initialbal;

                }

                void inquiry()

                {

                        cout<<"current balance:"<<bal<<endl;

                }

                void deposite(float amt)

                {

                        if(amt>0)

                        {

                                bal+=amt;

                                cout<<"amount deposited:"<<amt<<endl;

                                cout<<"Current Balance:"<<bal<<endl;

                        }
                        else
                        {
                                cout<<"invalid amount\n";
                        }
                }
                void withdraw(float amt)
                {
                   if(amt>bal)
                   {
                        cout<<"Insufficient Balance"<<endl;
                   }
                   else
                   {
                        bal-=amt;
                        cout<<"Amount Withdraw:"<<amt<<endl;
                        cout<<"Current Balance:"<<bal<<endl;
                   }

        }
};
void main()
{
        int ch,b;
        float inbal;
        clrscr();
        cout<<"Enter initial bal."<<endl;
        cin>>inbal;
```

```
ATM a(inbal);
while(1)
{
        cout<<"1.Balance Enquiry."<<endl;
        cout<<"2.Deposite Balance."<<endl;
        cout<<"3.Withdraw Balance."<<endl;
        cout<<"4.Exit"<<endl;
        cout<<"Enter Your Choice:";
        cin>>ch;
        switch(ch)
        {
                case 1:
                        a.inquiry();
                        break;
                case 2:
                        cout<<"Enter Ammont to Deposite:";
                        cin>>b;
                        a.deposite(b);
                        break;
                case 3:
                        cout<<"Enter Amount to Withdrawal:";
                        cin>>b;
                        a.withdraw(b);
                        break;
                case 4:
                        cout<<"Exiting";
                        return;
```

```
                                break;
                        default:
                                cout<<"Invalid Choice";
                                break;
                }
        }
        getch();
}
```

Output:

```
1.Balance Enquiry.
2.Deposite Balance.
3.Withdraw Balance.
4.Exit
Enter Your Choice:1
current balance:1500
1.Balance Enquiry.
2.Deposite Balance.
3.Withdraw Balance.
4.Exit
Enter Your Choice:2
Enter Amount to Deposite:1500
amount deposited:1500
Current Balance:3000
1.Balance Enquiry.
2.Deposite Balance.
3.Withdraw Balance.
```

```
4.Exit

Enter Your Choice:3

Enter Amount to Withdrawal:1000

Amount Withdraw:1000

Current Balance:2000

1.Balance Enquiry.

2.Deposite Balance.

3.Withdraw Balance.

4.Exit

Enter Your Choice:4

Exiting
```

Assignment 2

Create a C++ program to simulate a basic calculator supporting expression parsing using function overloading and inline functions.

Program:

```
#include <iostream.h>

#include <conio.h>

#include <stdlib.h>


inline int add(int a, int b) { return a + b; }
inline int subtract(int a, int b) { return a - b; }
inline int multiply(int a, int b) { return a * b; }
inline int divide(int a, int b) { return a / b; }

inline float add(float a, float b) { return a + b; }
inline float subtract(float a, float b) { return a - b; }
inline float multiply(float a, float b) { return a * b; }
inline float divide(float a, float b) { return a / b; }

void evaluateExpression() {
  float num1, num2;
  char op;

  cout << "Enter an expression (e.g., 3 + 4 or 3.5 * 2): ";
  cin >> num1 >> op >> num2;

  if(num1 == (int)num1 && num2 == (int)num2) {
    int n1 = (int)num1;
    int n2 = (int)num2;
    switch(op) {
      case '+': cout << "Result = " << add(n1, n2); break;
      case '-': cout << "Result = " << subtract(n1, n2); break;
      case '*': cout << "Result = " << multiply(n1, n2); break;
      case '/':
        if(n2 != 0) cout << "Result = " << divide(n1, n2);
        else cout << "Error: Division by zero!";
        break;
```

```
            default: cout << "Invalid operator!";
        }
    } else {
        switch(op) {
            case '+': cout << "Result = " << add(num1, num2); break;
            case '-': cout << "Result = " << subtract(num1, num2); break;
            case '*': cout << "Result = " << multiply(num1, num2); break;
            case '/':
                if(num2 != 0) cout << "Result = " << divide(num1, num2);
                else cout << "Error: Division by zero!";
                break;
            default: cout << "Invalid operator!";
        }
    }
    cout << "\n";
}

void main() {
    clrscr();
    char choice;
    do {
        evaluateExpression();
        cout << "Do you want to calculate again? (y/n): ";
        cin >> choice;
    } while(choice == 'y' || choice == 'Y');
    getch();
}
```

Output:

```
Enter an expression (e.g., 3 + 4 or 3.5 * 2): 6+7

Result = 13

Do you want to calculate again? (y/n): y

Enter an expression (e.g., 3 + 4 or 3.5 * 2): 3.4*4.6

Result = 15.64

Do you want to calculate again? (y/n): n
```

Assignment No-3

Menu-driven C++ program that simulates a student grading system using control structures (if-else, switch, loops).

```
#include <iostream.h>

char calculateGrade(float marks) {

   if (marks >= 90) return 'A';

   else if (marks >= 80) return 'B';

   else if (marks >= 70) return 'C';

   else if (marks >= 60) return 'D';

   else if (marks >= 50) return 'E';

   else return 'F';

}


int main() {

   int choice;

   int numStudents;
```

```
float marks;

do {
    cout << "\n===== Student Grading System =====\n";
    cout << "1. Enter marks of a single student\n";
    cout << "2. Enter marks of multiple students\n";
    cout << "3. Exit\n";
    cout << "Enter your choice: ";
    cin >> choice;

    switch(choice) {
        case 1:
            cout << "Enter marks (0-100): ";
            cin >> marks;
            if (marks < 0 || marks > 100) {
                cout << "Invalid marks entered!" << endl;
            } else {
                cout << "Grade: " << calculateGrade(marks) << endl;
            }
            break;

        case 2:
            cout << "Enter number of students: ";
            cin >> numStudents;
            for (int i = 1; i <= numStudents; i++) {
                cout << "Enter marks of student " << i << ": ";
                cin >> marks;
                if (marks < 0 || marks > 100) {
                    cout << "Invalid marks entered for student " << i << endl;
                } else {
                    cout << "Grade of student " << i << ": "
                         << calculateGrade(marks) << endl;
                }
            }
            break;

        case 3:
            cout << "Exiting... Thank you!\n";
            break;

        default:
            cout << "Invalid choice. Please try again.\n";
    }

} while(choice != 3);

return 0;
```

```
}
```

Assignment 4

Implement a class for a Complex Number,overload the +,-,and * operators,and manage memory using dynamic allocation.

Program:

```
#include <iostream.h>
#include <conio.h>
#include <stdlib.h>

class Complex {
private:
   float real;
   float imag;

public:

   Complex(float r = 0, float i = 0) {
      real = r;
      imag = i;
   }


   Complex operator+(Complex c) {
      return Complex(real + c.real, imag + c.imag);
   }


   Complex operator-(Complex c) {
      return Complex(real - c.real, imag - c.imag);
   }


   Complex operator*(Complex c) {
      return Complex(real * c.real - imag * c.imag,real * c.imag + imag * c.real);
   }

   void display() {
      cout << real;
      if (imag >= 0)
         cout << " + " << imag << "i";
      else
         cout << " - " << -imag << "i";
```

```
        cout << endl;
    }
};

int main() {
    clrscr();

    Complex *c1, *c2, *c3;


    c1 = new Complex(3, 2);
    c2 = new Complex(1, 7);

    cout << "First Complex Number: ";
    c1->display();

    cout << "Second Complex Number: ";
    c2->display();

    c3 = new Complex(*c1 + *c2);
    cout << "\nAddition: ";
    c3->display();
    delete c3;

    c3 = new Complex(*c1 - *c2);
    cout << "Subtraction: ";
    c3->display();
    delete c3;

    c3 = new Complex(*c1 * *c2);
    cout << "Multiplication: ";
    c3->display();
    delete c3;


    delete c1;
    delete c2;

    getch();
    return 0;
}
```

Output:

```
First Complex Number: 3 + 2i
Second Complex Number: 1 + 7i

Addition: 4 + 9i
Subtraction: 2 - 5i
Multiplication: -11 + 23i
```

Assignment 5

Design a class for a Student Database that allows the addition,deletion,and search of student records using static members and friend functions

Program:

```
#include <iostream.h>
#include <conio.h>
#include <string.h>

class Student;


void searchStudent(Student *s, int n, int r);
void deleteStudent(Student *s, int &n, int r);

class Student {
private:
    int roll;
    char name[30];

public:
    static int count;

    void getData() {
        cout << "Enter Roll No: ";
        cin >> roll;
        cout << "Enter Name: ";
        cin >> name;
        count++;
    }

    void showData() {
        cout << roll << "\t" << name << endl;
    }


    friend void searchStudent(Student *s, int n, int r);
    friend void deleteStudent(Student *s, int &n, int r);
};

int Student::count = 0;


void searchStudent(Student *s, int n, int r) {
    for (int i = 0; i < n; i++) {
        if (s[i].roll == r) {
            cout << "Record Found\n";
            cout << "Roll\tName\n";
            s[i].showData();
            return;
        }
    }
    cout << "Record Not Found\n";
}


void deleteStudent(Student *s, int &n, int r) {
    for (int i = 0; i < n; i++) {
        if (s[i].roll == r) {
            for (int j = i; j < n - 1; j++) {
                s[j] = s[j + 1];
            }
            n--;
            Student::count--;
            cout << "Record Deleted\n";
            return;
        }
    }
    cout << "Record Not Found\n";
}

int main() {
    clrscr();
    Student s[20];
    int n = 0, choice, roll;

    do {
            cout << "\n1.Add \n2.Display \n3.Search \n4.Delete \n5.Exit\n";
            cin >> choice;

            switch (choice) {
                case 1:
                        s[n].getData();
                        n++;
                        break;

                case 2:
                        for (int i = 0; i < n; i++)
                            s[i].showData();
                        break;

                case 3:
                        cout << "\nEnter Roll to Search: ";
                        cin >> roll;
                        searchStudent(s, n, roll);
                        break;

                case 4:
                        cout << "\nEnter Roll to Delete: ";
                cin >> roll;
                deleteStudent(s, n, roll);
                break;
        }
    } while (choice != 5);

    getch();
    return 0;
}
```

Output:

```
1.Add
2.Display
3.Search
4.Delete
5.Exit
1
Enter Roll No: 65
Enter Name: rudraksh

1.Add
2.Display
3.Search
4.Delete
5.Exit
1
Enter Roll No: 4
Enter Name: Bharat

1.Add
2.Display
3.Search
4.Delete
5.Exit
1
Enter Roll No: 64
Enter Name: Manish

1.Add
2.Display
3.Search
4.Delete
```

```
5.Exit
2
65      rudraksh
4       Bharat
64      Manish

1.Add
2.Display
3.Search
4.Delete
5.Exit
3

Enter Roll to Search: 64
Record Found
Roll    Name
64      Manish

1.Add
2.Display
3.Search
4.Delete
5.Exit
4

Enter Roll to Delete: 65
Record Deleted

1.Add
2.Display
3.Search
4.Delete
5.Exit
2
4       Bharat
64      Manish

1.Add
2.Display
3.Search
4.Delete
5.Exit
```

Assignment 6

Create a pointer-based implementation of a class where objects are dynamically created and manipulated using the this pointer and constructors/destructors.

Program:

```
#include <iostream.h>
#include <conio.h>

class Student {
private:
   int roll;
   float marks;

public:
      Student(int roll, float marks) {
      this->roll = roll;
      this->marks = marks;
      cout << "Object Created\n";
   }

   void show() {
      cout << "Roll: " << roll << endl;
      cout << "Marks: " << marks << endl;
   }


   ~Student() {
      cout << "Object Destroyed\n";
   }
};

int main() {
   clrscr();
   Student *s;
   s = new Student(101, 85.5);
   s->show();
   delete s;
   getch();
   return 0;
}
```

Output:

```
Object Created
```

```
Roll: 101
Marks: 85.5
Object Destroyed
```

Assignment 7

Implement a multi-level inheritance model representing an organization hierarchy and demonstrate method overriding and virtual functions

Program:

```
#include <iostream.h>
#include <conio.h>
#include<string.h>
class Employee {
protected:
   int id;
public:
   Employee(int i) {
      id = i;
   }

   virtual void display() {
      cout << "Employee ID: " << id << endl;
   }
};

class Manager : public Employee {
protected:
   char department[20];
public:
   Manager(int i, char d[]) : Employee(i) {
      strcpy(department, d);
   }

   void display() {
      cout << "Manager ID: " << id << endl;
      cout << "Department: " << department << endl;
   }
};
```

```
class Director : public Manager {
protected:
    float budget;
public:
    Director(int i, char d[], float b) : Manager(i, d) {
        budget = b;
    }

    void display() {
        cout << "Director ID: " << id << endl;
        cout << "Department: " << department << endl;
        cout << "Budget: " << budget << endl;
    }
};

int main() {
    clrscr();

    Employee *e;
    Director d(101, "IT", 500000);

    e = &d;

    e->display();

    getch();
    return 0;
}
```

Output:

```
Director ID: 101
Department: IT
Budget: 500000
```

Assignment 8

Create a program using hybrid inheritance to model academic and extracurricular performance and resolve ambiguity using virtual base classes.

Program:

```
#include <iostream.h>
#include <conio.h>
#include <string.h>


class Student {
protected:
    int rollNo;
public:
    Student(int r) {
        rollNo = r;
    }
    void showRoll() {
        cout << "Roll Number: " << rollNo << endl;
    }
};


class Academic : virtual public Student {
protected:
    int marks;
public:
    Academic(int r, int m) : Student(r) {
        marks = m;
    }
    void showAcademic() {
        cout << "Academic Marks: " << marks << endl;
    }
};


class Extracurricular : virtual public Student {
protected:
    char activity[50];
public:
    Extracurricular(int r, char a[]) : Student(r) {
        strcpy(activity, a);
    }
    void showActivity() {
        cout << "Extracurricular Activity: " << activity << endl;
    }
};


class Performance : public Academic, public Extracurricular {
public:
    Performance(int r, int m, char a[])
        : Student(r), Academic(r, m), Extracurricular(r, a) { }

    void showPerformance() {
        showRoll();
        showAcademic();
        showActivity();
    }
};

int main() {
    clrscr();
    Performance p(101, 95, "Cricket");
    p.showPerformance();
    getch();
    return 0;
}
```

Output:

```
Roll Number: 101
Academic Marks: 95
Extracurricular Activity: Cricket
```

Assignment 9

Design a payroll system using run-time polymorphism (abstract classes + purevirtual functions) for different employee types (hourly,salaried,contractual).

Program:

```
#include <iostream.h>
#include <conio.h>
#include<string.h>
class Employee {
protected:
    int empID;
    char name[30];
public:
    Employee(int id, char n[]) {
        empID = id;
        strcpy(name, n);
    }
    virtual float calculateSalary() = 0;
    void showDetails() {
        cout << "ID: " << empID << "\tName: " << name << endl;
    }
};


class HourlyEmployee : public Employee {
private:
    int hours;
    float rate;
public:
    HourlyEmployee(int id, char n[], int h, float r) : Employee(id, n) {
        hours = h;
        rate = r;
    }
    float calculateSalary() {
        return hours * rate;
    }
};

class SalariedEmployee : public Employee {
private:
    float salary;
public:
    SalariedEmployee(int id, char n[], float s) : Employee(id, n) {
        salary = s;
    }
    float calculateSalary() {
        return salary;
    }
};


class ContractualEmployee : public Employee {
private:
    int projects;
    float payPerProject;
public:
    ContractualEmployee(int id, char n[], int p, float pay) : Employee(id, n) {
        projects = p;
        payPerProject = pay;
    }
    float calculateSalary() {
        return projects * payPerProject;
    }
};

int main() {
    clrscr();

    Employee *e;

    HourlyEmployee h(101, "Rudraksh", 40, 200);
    SalariedEmployee s(102, "Ananya", 50000);
    ContractualEmployee c(103, "Kunal", 3, 15000);


    e = &h;
    e->showDetails();
    cout << "Salary: " << e->calculateSalary() << endl << endl;

    e = &s;
    e->showDetails();
    cout << "Salary: " << e->calculateSalary() << endl << endl;
```

```
   e = &c;
   e->showDetails();
   cout << "Salary: " << e->calculateSalary() << endl;

   getch();
   return 0;
}
```

Output:

```
ID: 101  Name: Rudraksh
Salary: 8000

ID: 102  Name: Ananya
Salary: 50000

ID: 103  Name: Kunal
Salary: 45000
```

Assignment 12

Write a C++ program to simulate a bank transaction system that throws and handles multiple exceptions(e.g.,insufficient funds,invalid account).

Program:

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

class InvalidAccount:public exception
{
   public:
      const char*what() const noexcept
      {
         return "Invalid Account Number!";
      }
};

class InsufficientFunds:public exception
{
   public:
      const char*what() const noexcept
      {
         return "Insufficient funds!";
```

```
        }
};

int main()
{
    int accountNumber;
    double balance=1000;
    double amount;

    try
    {
          cout<<"Enter Account number:";
          cin>>accountNumber;
        if(accountNumber!=1234)
        {
            throw InvalidAccount();

        }
        cout<<"Enter amount to withdraw:";
        cin>>amount;
        if(amount>balance)
        {
            throw InsufficientFunds();

        }
        balance-=amount;
        cout<<"Withdrawal Successful!New balance:"<<balance<<endl;

    }
    catch(exception &e)
    {
        cout<<"Exception:"<<e.what()<<endl;
    }
    return 0;
}
```

Output:

```
Enter Account number:123
Exception:Invalid Account Number!


Enter Account number:1234
Enter amount to withdraw:1600
Exception:Insufficient funds!
```

```
Enter Account number:1234
Enter amount to withdraw:500
Withdrawal Successful!New balance:500
```