Case Study 1

```cpp
// Case Study 1: Smart Calculator (Function Overloading)
// A software company is developing a Smart Calculator that supports different types of
// addition operations.
// You are asked to design a class Calculator with overloaded functions add() to handle different
// cases of addition:
// 1. Add two integers â†’ add(int a, int b)
// 2. Add three integers â†’ add(int a, int b, int c)
// 3. Add two floating-point numbers â†’ add(float x, float y)


# include <bits/stdc++.h>
using namespace std;

class calculator{

    public:


    int add(int n1,int n2){
        return (n1+n2);
    }
    double add(double n1,double n2){
        return (n1+n2);
    }
    int add(int n1,int n2,int n3){
        return (n1+n2+n3);
    }
};

int main(){
    calculator c1;
    int result1=c1.add(4,5);
    int result2=c1.add(4,5,6);
    double result3=c1.add(4.4,5.5);

    cout<<result1<<endl;
    cout<<result2<<endl;
    cout<<result3<<endl;


    return 0;
}
```

Case Study 2

```cpp
// Case Study 2: Volume Calculation System
// An engineering design application needs a module that can calculate volumes of different 3D
// shapes.
// You are asked to design a class Volume with overloaded functions compute() to handle
// different cases of volume calculation:
// 1. Volume of a cube â†’ compute(double side)
// 2. Volume of a cuboid â†’ compute(double length, double breadth, double height)
// 3. Volume of a cylinder â†’ compute(double radius, double height)


# include <bits/stdc++.h>
using namespace std;

class volume{

    public:
    void find_volume(double side){
        cout<<"the volume of cube is "<<pow(side,3)<<endl;
    }

    void find_volume(double length, double breadth, double height){
        cout<<"the volume of the cuboid is "<<length*breadth*height<<endl;
    }

    void find_volume(double radius,double height){
        cout<<"the volume of cylinder is "<<3.14*radius*radius*height<<endl;
    }

};

int main(){
    volume v1;
    v1.find_volume(3);
    v1.find_volume(3,4,5);
    v1.find_volume(3,4);

    return 0;
}
```

Case Study 3

```cpp
// Case Study 3: Book Information Access (Const Object)
// ï¸· A digital library system maintains book information where some details are read-only.
// ï¸· Define a class Book with attributes:
// o string title
// o string author
// ï¸· Add:
// o getTitle() as a const function (read-only).
// o setTitle() as a modifying function.
// ï¸· Create a const object of Book and demonstrate that it can only call getTitle() but not
// setTitle().

# include <bits/stdc++.h>
using namespace std;

class Book{
    string title;
    string author;

    public:
    Book(string t,string a){
        title=t;
        author=a;

    }


    void get_title()const {
        cout<<"title of the book is "<<title<<endl;
        cout<<"the author of the book is "<<author<<endl;
    }

    void set_title(string new_title,string new_author){
        title= new_title;
        author= new_author;

    }
};

int main(){
    cout<<"for normal object  --------------------------"<<endl;
    Book b1("Harry Potter and prisoner of askaban","JK Rowling");
    b1.get_title();

    b1.set_title("harry Potter","JK");
    b1.get_title();

    cout<<"for constant object -------------------------"<<endl;
    const Book b2("RD Sharma","Sharma JI");
    b2.get_title();
    // b2.set_title("RS aggarwal","Aggarwal ji");
    // this line would have worked if the object were not constant it won't compile as it is not possible to change a const onject



    return 0;
}
```

Case Study 4

```cpp
// Case Study 4: Library Book Tracking (Static Data Member)
// ï¿· A library maintains the total number of books issued and returned.
// ï¿· Define a class Library with a static data member totalBooks.
// ï¿· Add functions:
// o issueBook() â†’ decreases totalBooks
// o returnBook() â†’ increases totalBooks
// o showTotalBooks() â†’ displays the current count
// ï¿· Demonstrate issuing and returning books using multiple objects, and show that the
// static member is shared across all objects.

# include <bits/stdc++.h>
using namespace std;

class Library{
    static int total_number_of_books;
    string name_of_student;

    public:
    Library(string n){
    name_of_student=n; // name of the person whom the book got issued
    }



    void issueBook(){
        --total_number_of_books;
        cout<<name_of_student<<" issued a book"<<endl;
        cout<<"number of books are after issuing "<<total_number_of_books<<endl;
    }
    void returnBook(){
        ++total_number_of_books;
        cout<<name_of_student<<"returned a book"<<endl;
        cout<<"number of books after returning are "<<total_number_of_books<<endl;
    }
    void static showBook(){
        cout<<"the current number of books are "<<total_number_of_books<<endl;
    }

};

int Library::total_number_of_books=100;// defining the initial number of books to be 100 and

int main(){
    Library l1("amogh");
    l1.issueBook();
    l1.returnBook();
    l1.showBook();

    Library l2("areeb");
    l2.issueBook();
    l2.returnBook();
    l2.showBook();

    Library::showBook();



    return 0;
}
```

Case Study 5

```cpp
// Case Study 5: Geometry Assistant â€“ Perimeter Calculation Using Friend Function
// A math learning application needs a feature to calculate the perimeter of different geometrical
// figures.
// To implement this, you are asked to design separate classes for Rectangle, Circle, and
// Triangle.
// Since perimeter calculation often requires access to the figureâ€™s private data (like side
// lengths, radius, etc.), a friend function will be used.
// Requirements:
// 1. Create the following classes:
// o Rectangle â†’ with private members length and breadth.
// o Circle â†’ with private member radius.
// o Triangle â†’ with private members a, b, c (sides).
// 2. Define a friend function calculatePerimeter() that can access the private members of
// each class and compute their perimeters.

#include <bits/stdc++.h>
using namespace std;

class Rectangle {
    int length;
    int breadth;

public:
    Rectangle(int l, int b) {
        length = l;
        breadth = b;
    }

    // declare friend
    friend void calculatePerimeter(const Rectangle&);
};

class Circle {
    int radius;

public:
    Circle(int r) {
        radius = r;
    }

    // declare friend
    friend void calculatePerimeter(const Circle&);
};

class Triangle {
    int side1;
    int side2;
    int side3;

public:
    Triangle(int s1, int s2, int s3) {
        side1 = s1;
        side2 = s2;
        side3 = s3;
    }

    // declare friend
    friend void calculatePerimeter(const Triangle&);
};

// Friend function overloads
void calculatePerimeter(const Rectangle& r) {
    cout << "Perimeter of Rectangle: " << 2 * (r.length + r.breadth) << endl;
}

void calculatePerimeter(const Circle& c) {
    cout << "Perimeter of Circle: " << 2 * 3.14 * c.radius << endl;
}

void calculatePerimeter(const Triangle& t) {
    cout << "Perimeter of Triangle: " << t.side1 + t.side2 + t.side3 << endl;
}

int main() {
    Rectangle rect(10, 5);
    Circle cir(7);
    Triangle tri(3, 4, 5);

    calculatePerimeter(rect);
    calculatePerimeter(cir);
    calculatePerimeter(tri);

    return 0;
}
```

Case Study 6

```cpp
// Case Study 6: Bank Account Operations (Operator Overloading)
// A banking software system wants to make account transactions easier using operator
// overloading.
// Requirements
// 1. Define a class BankAccount with attributes:
// o accountNumber (int)
// o balance (double)
// 2. Overload the following operators:
// o + â†’ to deposit an amount to the account.
// o - â†’ to withdraw an amount from the account.
// o &lt;&lt; (stream insertion) â†’ to display account details.
// 3. In main(), create an object of BankAccount and perform deposit and withdrawal
// operations using the overloaded operators.

#include <bits/stdc++.h>
using namespace std;

class BankAccount {
    int accountNumber;
    double balance;

public:
    // constructor
    BankAccount(int accNo, double bal = 0.0) {
        accountNumber = accNo;
        balance = bal;
    }

    // overload + operator (deposit money)
    BankAccount operator+(double amount) {
        balance += amount;
        return *this; // return updated object
    }

    // overload - operator (withdraw money)
    BankAccount operator-(double amount) {
        if (amount <= balance) {
            balance -= amount;
        } else {
            cout << "Insufficient funds! Withdrawal not allowed." << endl;
        }
        return *this; // return updated object
    }

    // normal display function (instead of operator<<)
    void display() const {
        cout << "Account Number: " << accountNumber
             << " | Balance: " << balance << endl;
    }
};

int main() {
    BankAccount acc1(101, 5000);  // account with initial balance 5000

    cout << "Initial Account Details: ";
    acc1.display();

    acc1 = acc1 + 2000;  // deposit
    cout << "After deposit: ";
    acc1.display();

    acc1 = acc1 - 3000;  // withdraw
    cout << "After withdrawal: ";
    acc1.display();

    acc1 = acc1 - 6000;  // trying to withdraw more than balance
    cout << "After failed withdrawal: ";
    acc1.display();

    return 0;
}
```

Case Study 7

```cpp
// Case Study 7: University System (Inheritance)
// A university needs a software module to manage students and teachers. This can be designed
// using inheritance.
// 1. Create a base class Person with attributes:
// o name (string)
// o age (int)
// 2. Derive two classes from Person:
// o Student â†’ with additional attributes rollNumber and course.
// o Teacher â†’ with additional attributes employeeId and subject.
// 3. Both classes should have functions to input details and display details.
// 4. Demonstrate inheritance by creating objects of Student and Teacher and calling their
// respective methods.

#include <bits/stdc++.h>
using namespace std;

// Base class
class Person {
protected:
    string name;
    int age;

public:
    void inputPerson() {
        cout << "Enter name: ";
        getline(cin, name);
        cout << "Enter age: ";
        cin >> age;
        cin.ignore(); // clear input buffer
    }

    void displayPerson() const {
        cout << "Name: " << name << ", Age: " << age << endl;
    }
};

// Derived class: Student
class Student : public Person {
    int rollNumber;
    string course;

public:
    void inputStudent() {
        inputPerson(); // call base class function
        cout << "Enter roll number: ";
        cin >> rollNumber;
        cin.ignore();
        cout << "Enter course: ";
        getline(cin, course);
    }

    void displayStudent() const {
        displayPerson(); // call base class display
        cout << "Roll Number: " << rollNumber
             << ", Course: " << course << endl;
    }
};

// Derived class: Teacher
class Teacher : public Person {
    int employeeId;
    string subject;

public:
    void inputTeacher() {
        inputPerson(); // call base class function
        cout << "Enter employee ID: ";
        cin >> employeeId;
        cin.ignore();
        cout << "Enter subject: ";
        getline(cin, subject);
    }

    void displayTeacher() const {
        displayPerson(); // call base class display
        cout << "Employee ID: " << employeeId
             << ", Subject: " << subject << endl;
    }
};

int main() {
    cout << "--- Enter Student Details ---" << endl;
    Student s;
    s.inputStudent();

    cout << "\n--- Enter Teacher Details ---" << endl;
    Teacher t;
    t.inputTeacher();

    cout << "\n--- Student Information ---" << endl;
    s.displayStudent();

    cout << "\n--- Teacher Information ---" << endl;
    t.displayTeacher();

    return 0;
}
```