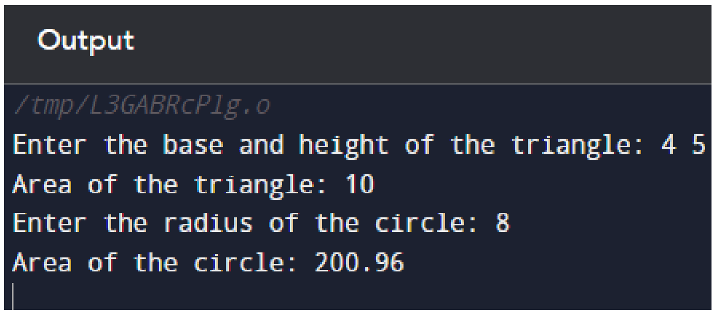
ASSIGNMENT NO:1

```
#include <iostream>

using namespace std;

class Area {
public:
    // Function to compute the area of a triangle
    float area(float base, float height) {
        return 0.5 * base * height;
    }

    // Function to compute the area of a circle
    float area(float radius) {
        return 3.14 * radius * radius;
    }
};

int main() {
    Area a;
    float base, height, radius;

    cout << "Enter the base and height of the triangle: ";
    cin >> base >> height;
    cout << "Area of the triangle: " << a.area(base, height) << endl;
    cout << "Enter the radius of the circle: ";
    cin >> radius;
    cout << "Area of the circle: " << a.area(radius) << endl;

    return 0;
}
```

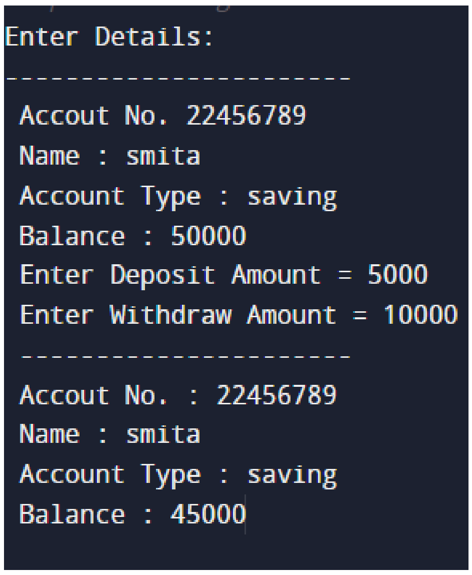

Output

```
/tmp/L3GABRcPlg.o
Enter the base and height of the triangle: 4 5
Area of the triangle: 10
Enter the radius of the circle: 8
Area of the circle: 200.96
```

ASSICNMENT NO:2

```
#include<iostream>

#include<stdio.h>

#include<string.h>


using namespace std;


class bank

{

        int acno;

        char nm[100], acctype[100];

        float bal;

   public:

        bank(int acc_no, char *name, char *acc_type, float balance)
//Parameterized Constructor

        {

                acno=acc_no;

                strcpy(nm, name);

                strcpy(acctype, acc_type);

                bal=balance;

        }

        void deposit();

        void withdraw();

        void display();

};

void bank::deposit()   //depositing an amount
```

```
{
        int damt1;
        cout<<"\n Enter Deposit Amount = ";
        cin>>damt1;
        bal+=damt1;
}
void bank::withdraw()  //withdrawing an amount
{
        int wamt1;cout<<"\n Enter Withdraw Amount = ";
        cin>>wamt1;
        if(wamt1>bal)
                cout<<"\n Cannot Withdraw Amount";
        bal-=wamt1;
}
void bank::display()  //displaying the details
{
        cout<<"\n ----------------------";
        cout<<"\n Accout No. : "<<acno;
        cout<<"\n Name : "<<nm;
        cout<<"\n Account Type : "<<acctype;
        cout<<"\n Balance : "<<bal;
}
int main()
{
        int acc_no;
        char name[100], acc_type[100];
        float balance;
        cout<<"\n Enter Details: \n";
        cout<<"-----------------------";
        cout<<"\n Accout No. ";
        cin>>acc_no;
        cout<<"\n Name : ";
        cin>>name;
        cout<<"\n Account Type : ";
        cin>>acc_type;
        cout<<"\n Balance : ";
        cin>>balance;

        bank b1(acc_no, name, acc_type, balance);  //object is created
        b1.deposit(); //
        b1.withdraw(); // calling member functions
        b1.display(); //
        return 0;
}
```

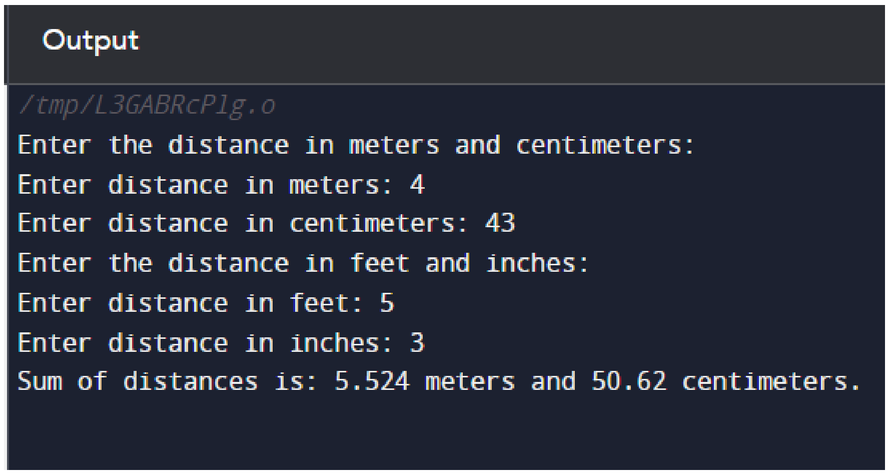


```
Enter Details:
-----------------------
 Accout No. 22456789
 Name : smita
 Account Type : saving
 Balance : 50000
 Enter Deposit Amount = 5000
 Enter Withdraw Amount = 10000
 ----------------------
 Accout No. : 22456789
 Name : smita
 Account Type : saving
 Balance : 45000
```

ASSIGNMENT NO:3

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

class DB; // Forward declaration

class DM {
private:
    int meters;
    float centimeters;

public:
    void getdata() {
        cout << "Enter distance in meters: ";
        cin >> meters;
        cout << "Enter distance in centimeters: ";
        cin >> centimeters;
    }

    friend void add(DM, DB);
};

class DB {
private:
    int feet;
    float inches;
```

```
public:
    void getdata() {
        cout << "Enter distance in feet: ";
        cin >> feet;
        cout << "Enter distance in inches: ";
        cin >> inches;
    }

    friend void add(DM, DB);
};

void add(DM dm, DB db) {
    float total_meters = dm.meters + db.feet * 0.3048;
    float total_centimeters = dm.centimeters + db.inches * 2.54;

    if (total_centimeters >= 100) {
        int extra_meters = total_centimeters / 100;
        total_meters += extra_meters;
        total_centimeters -= extra_meters * 100;
    }

    cout << "Sum of distances is: " << total_meters << " meters and " <<
total_centimeters << " centimeters." << endl;
}

int main() {
    DM dm;
    DB db;

    cout << "Enter the distance in meters and centimeters: " << endl;
    dm.getdata();

    cout << "Enter the distance in feet and inches: " << endl;
    db.getdata();

    add(dm, db);

    return 0;
}
```



Output

```
/tmp/L3GABRcPlg.o
Enter the distance in meters and centimeters:
Enter distance in meters: 4
Enter distance in centimeters: 43
Enter the distance in feet and inches:
Enter distance in feet: 5
Enter distance in inches: 3
Sum of distances is: 5.524 meters and 50.62 centimeters.
```

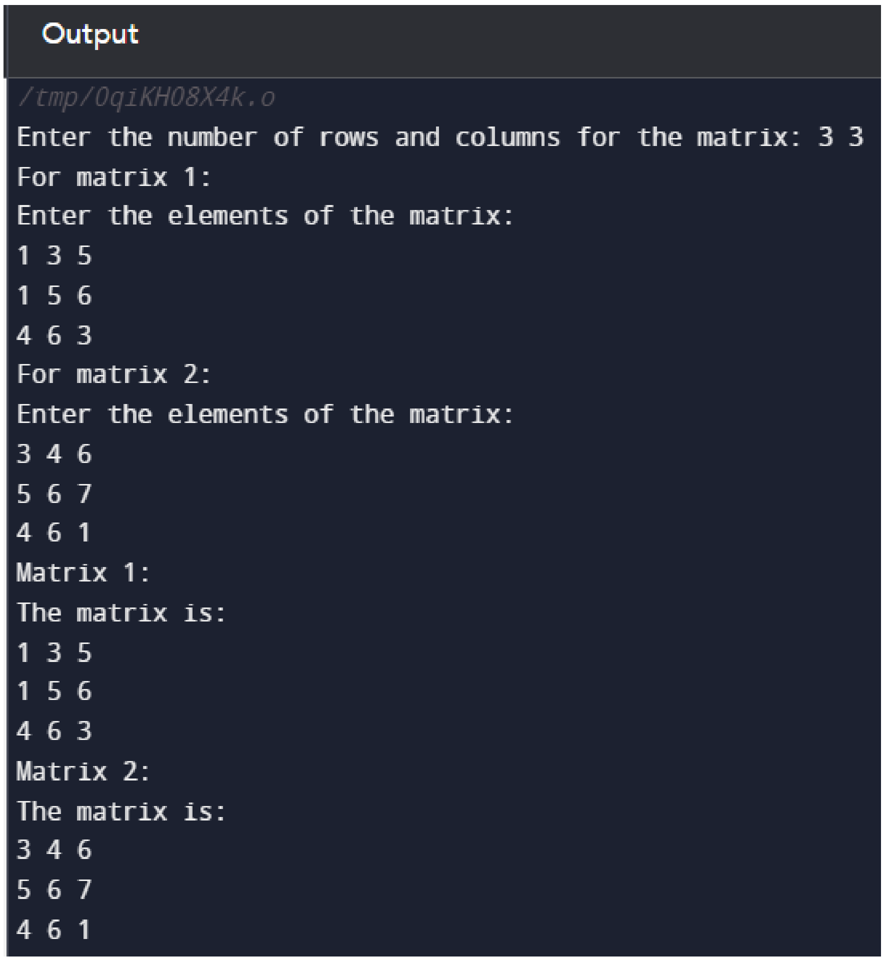
ASSIGNMENT NO: 4

```
#include <iostream>
#include <vector>

using namespace std;

class MAT {
private:
    vector<vector<int>> matrix;
    int m, n;

public:
    MAT(int m, int n) {
        this->m = m;
        this->n = n;
        matrix.resize(m, vector<int>(n, 0));
    }

    void inputMatrix() {
        cout << "Enter the elements of the matrix:" << endl;
        for (int i = 0; i < m; i++) {
            for (int j = 0; j < n; j++) {
                cin >> matrix[i][j];
            }
        }
    }
```

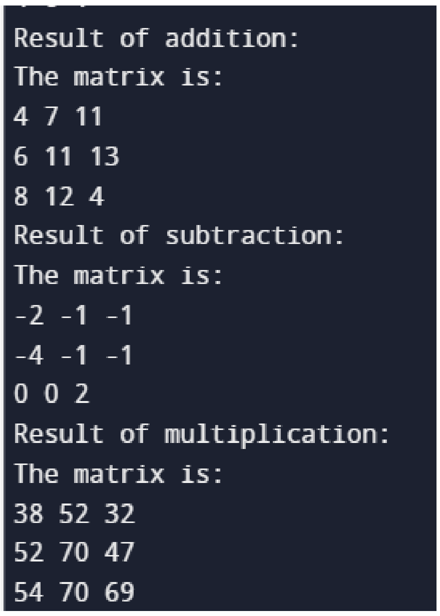
```
void displayMatrix() {
    cout << "The matrix is:" << endl;
    for (int i = 0; i < m; i++) {
        for (int j = 0; j < n; j++) {
            cout << matrix[i][j] << " ";
        }
        cout << endl;
    }
}

MAT add(MAT &other) {
    MAT result(m, n);
    for (int i = 0; i < m; i++) {
        for (int j = 0; j < n; j++) {
            result.matrix[i][j] = matrix[i][j] + other.matrix[i][j];
        }
    }
    return result;
}

MAT subtract(MAT &other) {
    MAT result(m, n);
    for (int i = 0; i < m; i++) {
        for (int j = 0; j < n; j++) {
            result.matrix[i][j] = matrix[i][j] - other.matrix[i][j];
            }
        }
        return result;
    }

    MAT multiply(MAT &other) {
        if (n != other.m) {
            throw "Invalid dimensions for matrix multiplication!";
        }
        MAT result(m, other.n);
        for (int i = 0; i < m; i++) {
            for (int j = 0; j < other.n; j++) {
                for (int k = 0; k < n; k++) {
                    result.matrix[i][j] += matrix[i][k] * other.matrix[k][j];
                }
            }
        }
        return result;
    }
};

int main() {
    int m, n;
    cout << "Enter the number of rows and columns for the matrix: ";
    cin >> m >> n;
```

```
MAT mat1(m, n), mat2(m, n);

cout << "For matrix 1:" << endl;
mat1.inputMatrix();

cout << "For matrix 2:" << endl;
mat2.inputMatrix();

// Display the matrices
cout << "Matrix 1:" << endl;
mat1.displayMatrix();
cout << "Matrix 2:" << endl;
mat2.displayMatrix();

// Perform matrix operations
MAT mat3 = mat1.add(mat2);
MAT mat4 = mat1.subtract(mat2);

MAT mat5 = mat1.multiply(mat2);

// Display the results
cout << "Result of addition:" << endl;
mat3.displayMatrix();
cout << "Result of subtraction:" << endl;
mat4.displayMatrix();
cout << "Result of multiplication:" << endl;
```

```
    mat5.displayMatrix();
return 0;
}
```



```
Output
/tmp/0qiKH08X4k.o
Enter the number of rows and columns for the matrix: 3 3
For matrix 1:
Enter the elements of the matrix:
1 3 5
1 5 6
4 6 3
For matrix 2:
Enter the elements of the matrix:
3 4 6
5 6 7
4 6 1
Matrix 1:
The matrix is:
1 3 5
1 5 6
4 6 3
Matrix 2:
The matrix is:
3 4 6
5 6 7
4 6 1
```

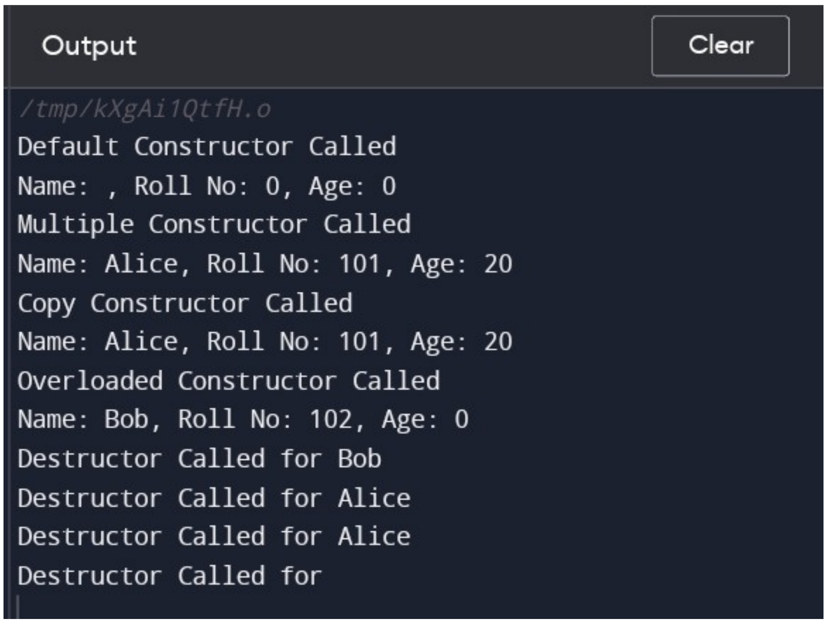


```
Result of addition:
The matrix is:
4 7 11
6 11 13
8 12 4
Result of subtraction:
The matrix is:
-2 -1 -1
-4 -1 -1
0 0 2
Result of multiplication:
The matrix is:
38 52 32
52 70 47
54 70 69
```

Program:-

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

class Stud {
private:
   std::string name;
   int rollNo;
   int age;

public:
   // Default Constructor
   Stud() : name(""), rollNo(0), age(0) {
      std::cout << "Default Constructor Called" << std::endl;
   }

   // Multiple Constructor
   Stud(std::string n, int r, int a) : name(n), rollNo(r), age(a) {
      std::cout << "Multiple Constructor Called" << std::endl;
   }

   // Copy Constructor
   Stud(const Stud& other) : name(other.name),
rollNo(other.rollNo), age(other.age) {
      std::cout << "Copy Constructor Called" << std::endl;
   }

   // Overloaded Constructor
   Stud(std::string n, int r) : name(n), rollNo(r), age(0) {
      std::cout << "Overloaded Constructor Called" << std::endl;
   }

   // Destructor
   ~Stud() {
      std::cout << "Destructor Called for " << name << std::endl;
   }
```

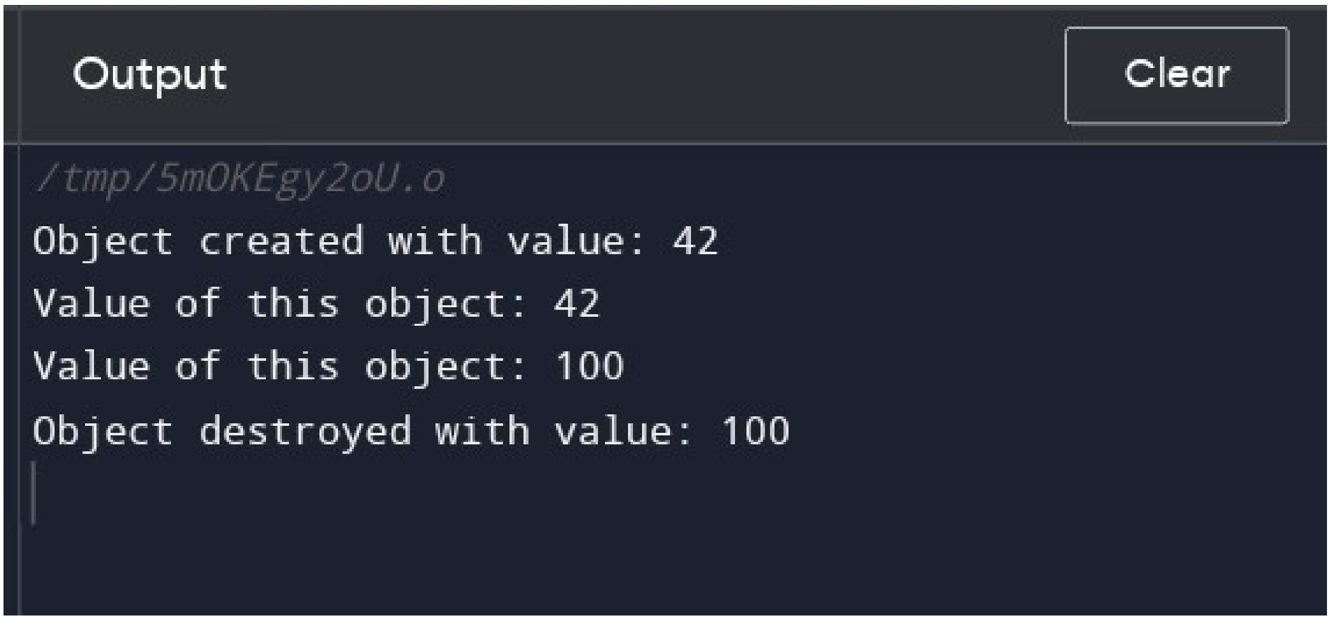
```
    void displayInfo() {
        std::cout << "Name: " << name << ", Roll No: " << rollNo << ", Age: " << age << std::endl;
    }
};

int main() {
    Stud student1; // Default Constructor
    student1.displayInfo();

    Stud student2("Alice", 101, 20); // Multiple Constructor
    student2.displayInfo();

    Stud student3 = student2; // Copy Constructor
    student3.displayInfo();

    Stud student4("Bob", 102); // Overloaded Constructor
    student4.displayInfo();

    return 0;
}
```



```
/tmp/kXgAi1QtfH.o
Default Constructor Called
Name: , Roll No: 0, Age: 0
Multiple Constructor Called
Name: Alice, Roll No: 101, Age: 20
Copy Constructor Called
Name: Alice, Roll No: 101, Age: 20
Overloaded Constructor Called
Name: Bob, Roll No: 102, Age: 0
Destructor Called for Bob
Destructor Called for Alice
Destructor Called for Alice
Destructor Called for
```

Program:-

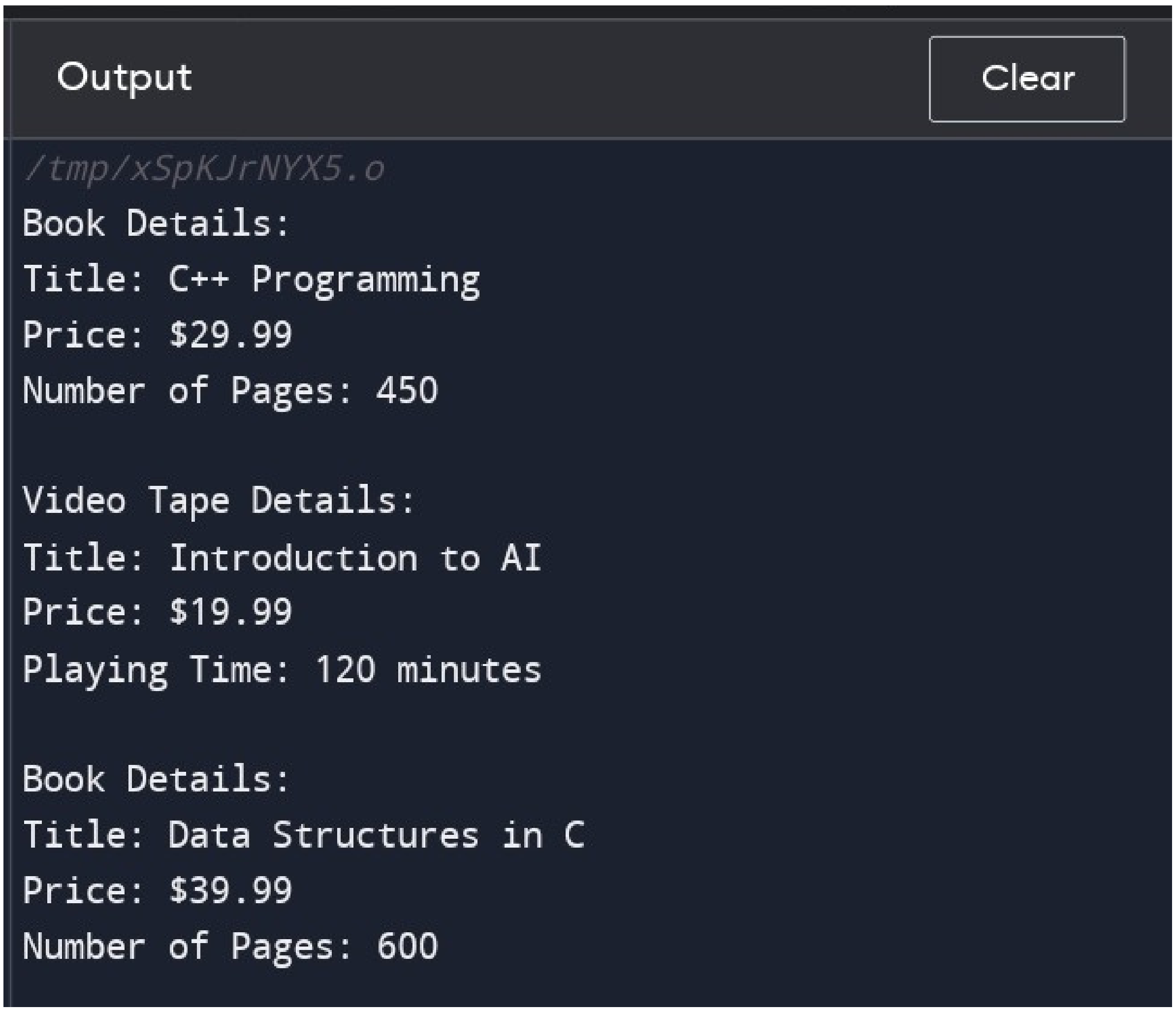
```
#include <iostream>

class MyClass {
public:
   MyClass(int value) : data(value) {
      std::cout << "Object created with value: " << data << std::endl;
   }

   void showValue() {
      std::cout << "Value of this object: " << this->data << std::endl;
   }

   void updateValue(int newValue) {
      this->data = newValue;
   }

   void releaseMemory() {
      delete this; // Delete the current object
   }

   ~MyClass() {
      std::cout << "Object destroyed with value: " << data << std::endl;
   }

private:
   int data;
};

int main() {
   MyClass* obj1 = new MyClass(42);

   obj1->showValue(); // Use this pointer to access member function

   obj1->updateValue(100);
   obj1->showValue();

   obj1->releaseMemory(); // Delete the object using the delete operator

   return 0;
}
```



```
/tmp/5mOKEgy2oU.o
Object created with value: 42
Value of this object: 42
Value of this object: 100
Object destroyed with value: 100
```

Program:-

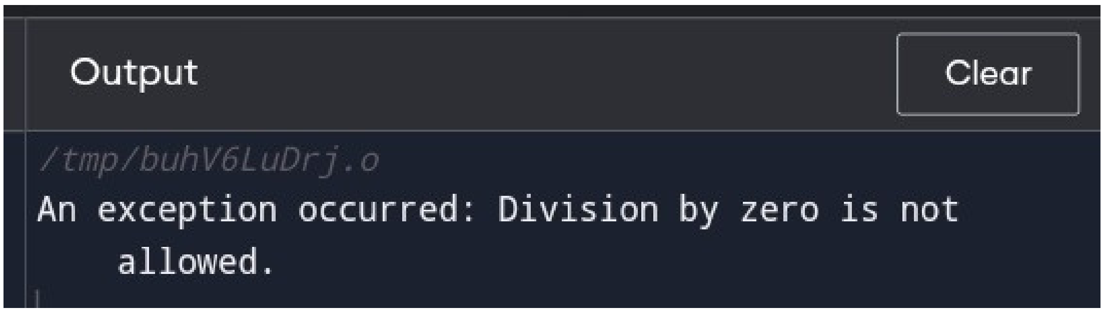
```
#include <iostream>
#include <string>

class Media {
protected:
   std::string title;
   double price;

public:
   Media(const std::string& t, double p) : title(t), price(p) {}

   virtual void display() {
      std::cout << "Title: " << title << "\nPrice: $" << price << std::endl;
   }
};

class Book : public Media {
   int numPages;

public:
   Book(const std::string& t, double p, int pages) : Media(t, p), numPages(pages) {}

   void display() override {
      std::cout << "Book Details:\n";
      Media::display();
      std::cout << "Number of Pages: " << numPages << std::endl;
   }
};

class VideoTape : public Media {
   int playTime;

public:
   VideoTape(const std::string& t, double p, int time) : Media(t, p), playTime(time) {}

   void display() override {
      std::cout << "Video Tape Details:\n";
      Media::display();
      std::cout << "Playing Time: " << playTime << " minutes" << std::endl;
   }
};

int main() {
   Media* items[3];

   items[0] = new Book("C++ Programming", 29.99, 450);
   items[1] = new VideoTape("Introduction to AI", 19.99, 120);
   items[2] = new Book("Data Structures in C", 39.99, 600);

   for (int i = 0; i < 3; i++) {
      items[i]->display();
      std::cout << std::endl;
      delete items[i];
   }

   return 0;
}
```



```
/tmp/xSpKJrNYX5.o
Book Details:
Title: C++ Programming
Price: $29.99
Number of Pages: 450

Video Tape Details:
Title: Introduction to AI
Price: $19.99
Playing Time: 120 minutes

Book Details:
Title: Data Structures in C
Price: $39.99
Number of Pages: 600
```

Program:-

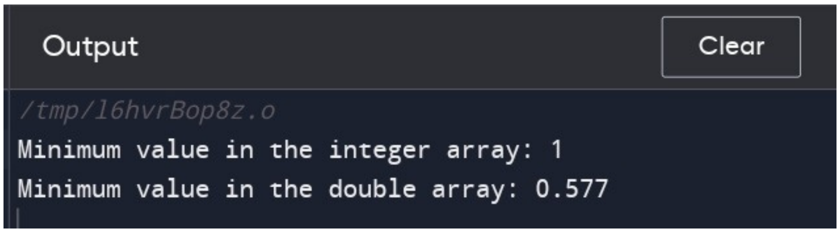
```
#include <iostream>

int main() {
    try {
        // Simulate a division by zero exception
        int numerator = 10;
        int denominator = 0;

        if (denominator == 0) {
            throw std::runtime_error("Division by zero is not
allowed.");
        }

        int result = numerator / denominator;
        std::cout << "Result: " << result << std::endl;
    } catch (std::exception &ex) {
        std::cerr << "An exception occurred: " << ex.what() <<
std::endl;
    }

    return 0;
}
```



```
/tmp/buhV6LuDrj.o
An exception occurred: Division by zero is not
    allowed.
```

Program:-

```
#include <iostream>

template <typename T>
T findMinimum(const T arr[], int size) {
   if (size <= 0) {
      // Handle empty array or invalid size
      std::cerr << "Error: Array is empty or size is invalid." << std::endl;
      return T();  // Return a default value for the type T
   }

   T min = arr[0];
   for (int i = 1; i < size; i++) {
      if (arr[i] < min) {
         min = arr[i];
      }
   }
   return min;
}

int main() {
   int intArr[] = {5, 2, 8, 1, 7};
   double doubleArr[] = {3.14, 2.71, 1.618, 0.577};

   int intMin = findMinimum(intArr, sizeof(intArr) / sizeof(intArr[0]));
   double doubleMin = findMinimum(doubleArr, sizeof(doubleArr) /
sizeof(doubleArr[0]));

   std::cout << "Minimum value in the integer array: " << intMin << std::endl;
   std::cout << "Minimum value in the double array: " << doubleMin << std::endl;

   return 0;
}
```

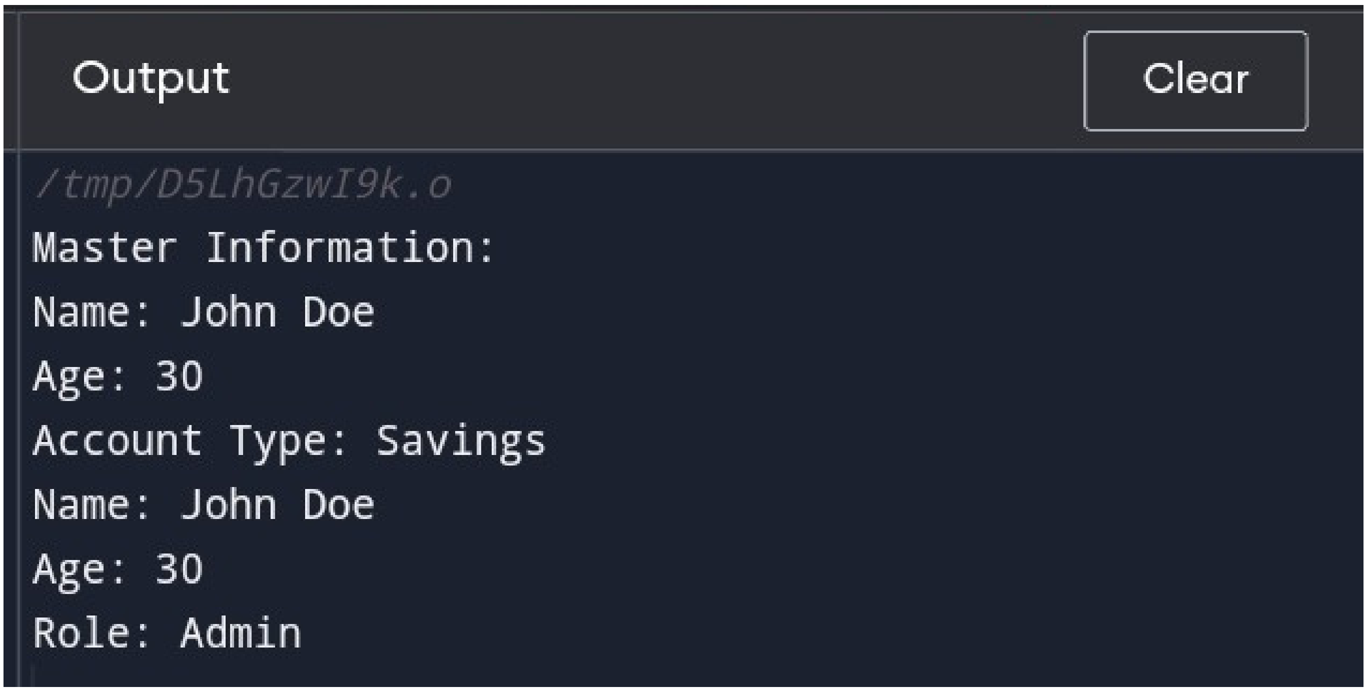


```
/tmp/l6hvrBop8z.o
Minimum value in the integer array: 1
Minimum value in the double array: 0.577
```

Program:-

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

class Person {
protected:
   std::string name;
   int age;

public:
   Person(const std::string& name, int age) : name(name), age(age) {}

   void display() {
      std::cout << "Name: " << name << "\nAge: " << age << std::endl;
   }
};

class Account : public Person {
protected:
   std::string accountType;

public:
   Account(const std::string& name, int age, const std::string& accountType)
: Person(name, age), accountType(accountType) {}

   void display() {
      Person::display();
      std::cout << "Account Type: " << accountType << std::endl;
   }
};

class Admin : public Person {
protected:
   std::string role;

public:
   Admin(const std::string& name, int age, const std::string& role) :
Person(name, age), role(role) {}
```

```
    void display() {
        Person::display();
        std::cout << "Role: " << role << std::endl;
    }
};

class Master : public Account, public Admin {
public:
    Master(const std::string& name, int age, const std::string&
accountType, const std::string& role)
        : Account(name, age, accountType), Admin(name, age, role) {}

    void display() {
        Account::display();
        Admin::display();
    }
};

int main() {
    Master master("John Doe", 30, "Savings", "Admin");

    // Display information from the master object
    std::cout << "Master Information:" << std::endl;
    master.display();

    // Update information in the master object
    master.name = "Jane Smith";
    master.age = 35;

    std::cout << "\nUpdated Master Information:" << std::endl;
    master.display();

    return 0;
}
```



```
/tmp/D5LhGzwI9k.o
Master Information:
Name: John Doe
Age: 30
Account Type: Savings
Name: John Doe
Age: 30
Role: Admin
```