1.Write a single C++ program :

(i) To find the square root of a number using a function. [Let the returntype of the function be void]

(ii) To increment a number using an inline function

(iii)To decrement a number using an inline function

**Program:**

#include <iostream>

#include <cmath>

void findSquareRoot(double number) {

double result = sqrt(number);

std::cout<<"The square root of"<<number<<"is"<<result<<std::endl;

}

inline void increment(int& number) {

++number;

}

inline void decrement(int& number) {

--number;

}

int main() {

double number;

int value;

std::cout << "Enter a number to find its square root: ";

std::cin >> number;

findSquareRoot(number);

std::cout << "Enter a number to increment: ";

std::cin >> value;

increment(value);

std::cout << "After incrementing: " << value << std::endl;

std::cout << "Enter a number to decrement: ";

std::cin >> value;

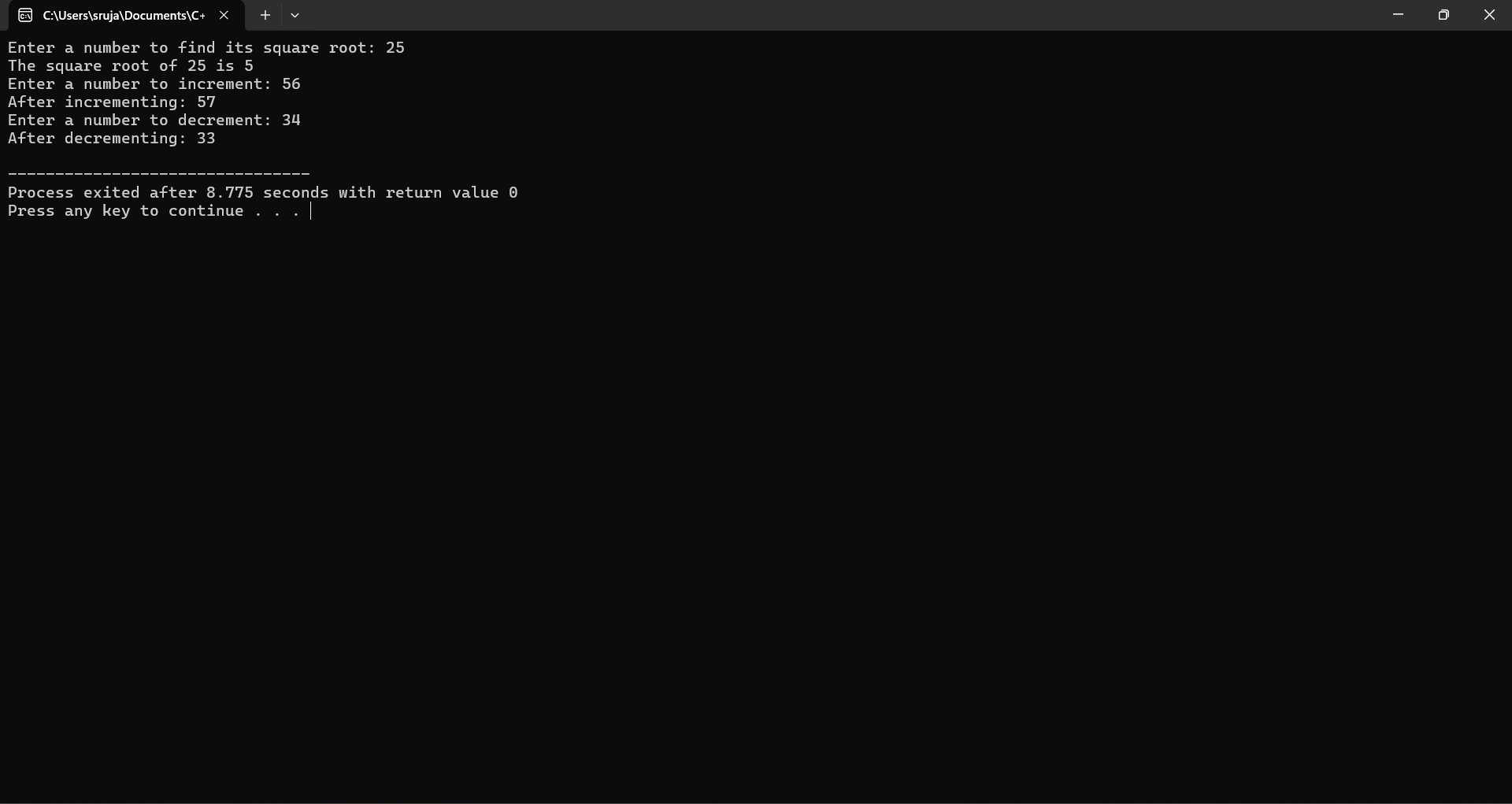
decrement(value);

std::cout << "After decrementing: " << value << std::endl;

return 0;

}

**OUTPUT:**

****

**2.** Create a class student:

\* Define a function getdata() and get the name,reg.no and 5 marks of a student.

[Note : 1.Use for loop for getting marks,2.Define the function inside the class]

\* Define a function tot\_marks() and calculate the total marks.

[Note: 1.Use for loop for calculating the total, 2.Define the function outside the class ]

**PROGRAM:**

#include <iostream>

#include <string>

using namespace std;

class Student {

string name;

string reg\_no;

int marks[5];

public:

void getdata() {

cout << "Enter name: ";

getline(cin, name);

cout << "Enter registration number: ";

getline(cin, reg\_no);

for (int i = 0; i < 5; i++) {

cout << "Enter mark " << i + 1 << ": ";

cin >> marks[i];

}

}

friend int tot\_marks(Student student);

};

int tot\_marks(Student student) {

int total = 0;

for (int i = 0; i < 5; i++) {

total += student.marks[i];

}

return total;

}

int main() {

Student student;

student.getdata();

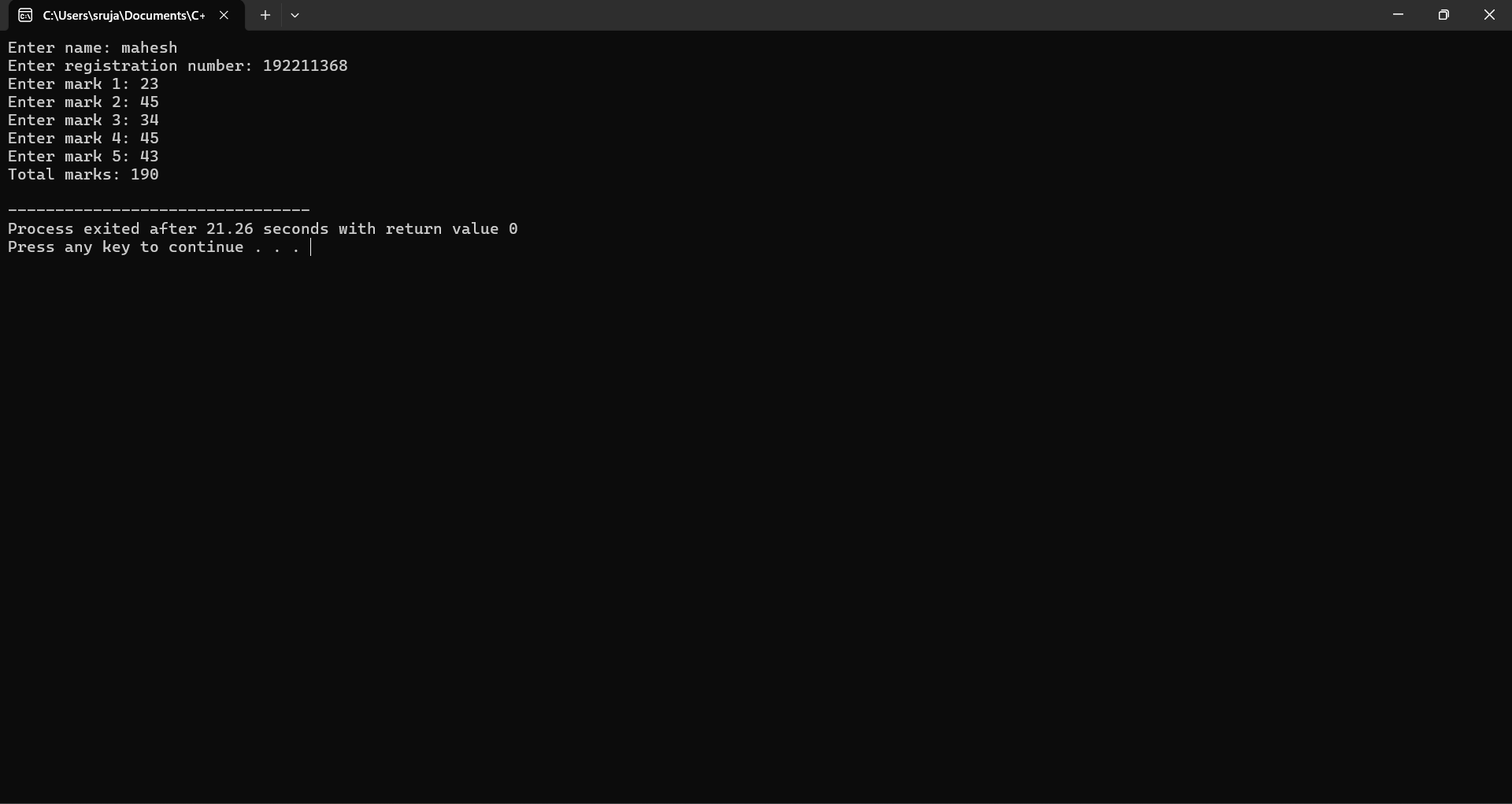
int total = tot\_marks(student);

cout << "Total marks: " << total << endl;

return 0;

}

**OUTPUT:**

****

**3.** Create a class product.

\* Define a function get\_product() and get the name of the product and its price.

\* Define a function print\_product() and display the product and its price.

\* Create an array of object to call the above functions.[Note: Array size: generalized]

PROGRAM:

**#include <iostream>**

**#include <vector>**

**using namespace std;**

**class Product {**

**string name;**

**double price;**

**public:**

**void get\_product() {**

**cout << "Enter product name: ";**

**cin >> name;**

**cout << "Enter product price: ";**

**cin >> price;**

**}**

**void print\_product() const {**

**cout << "Product Name: " << name << ", Price: " << price << endl;**

**}**

**};**

**int main() {**

**int n;**

**cout << "Enter the number of products: ";**

**cin >> n;**

**vector<Product> products(n);**

**for (int i = 0; i < n; ++i) {**

**cout << "Enter details for product " << i + 1 << ":" << endl;**

**products[i].get\_product();**

**}**

**cout << "\nProduct details:\n";**

**for (int i = 0; i < n; ++i) {**

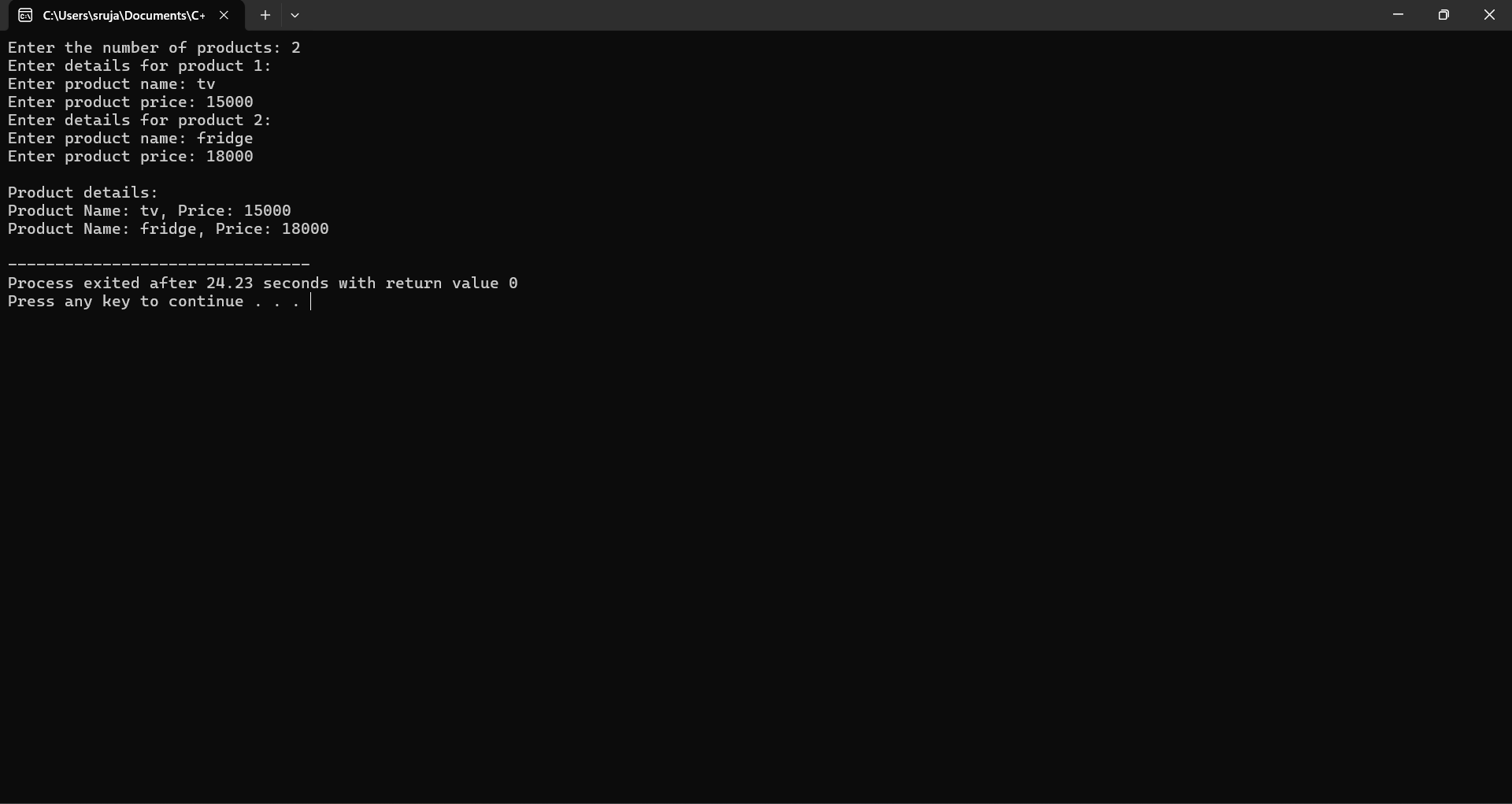
**products[i].print\_product();**

**}**

**return 0;**

**}**

**OUTPUT:**

****

**4.**

Write a C++ program to find the maximum of 2 numbers using a friend function.

\* Each number should be got in 2 different classes.

\* Define a friend function max which is common to both the class.

**PROGRAM:**#include <iostream>

using namespace std;

class ClassB;

class ClassA {

private:

int numA;

public:

ClassA(int n) : numA(n) {}

friend int max(ClassA, ClassB);

};

class ClassB {

private:

int numB;

public:

ClassB(int n) : numB(n) {}

friend int max(ClassA, ClassB);

};

int max(ClassA a, ClassB b) {

return (a.numA > b.numB) ? a.numA : b.numB;

}

int main() {

ClassA objA(10);

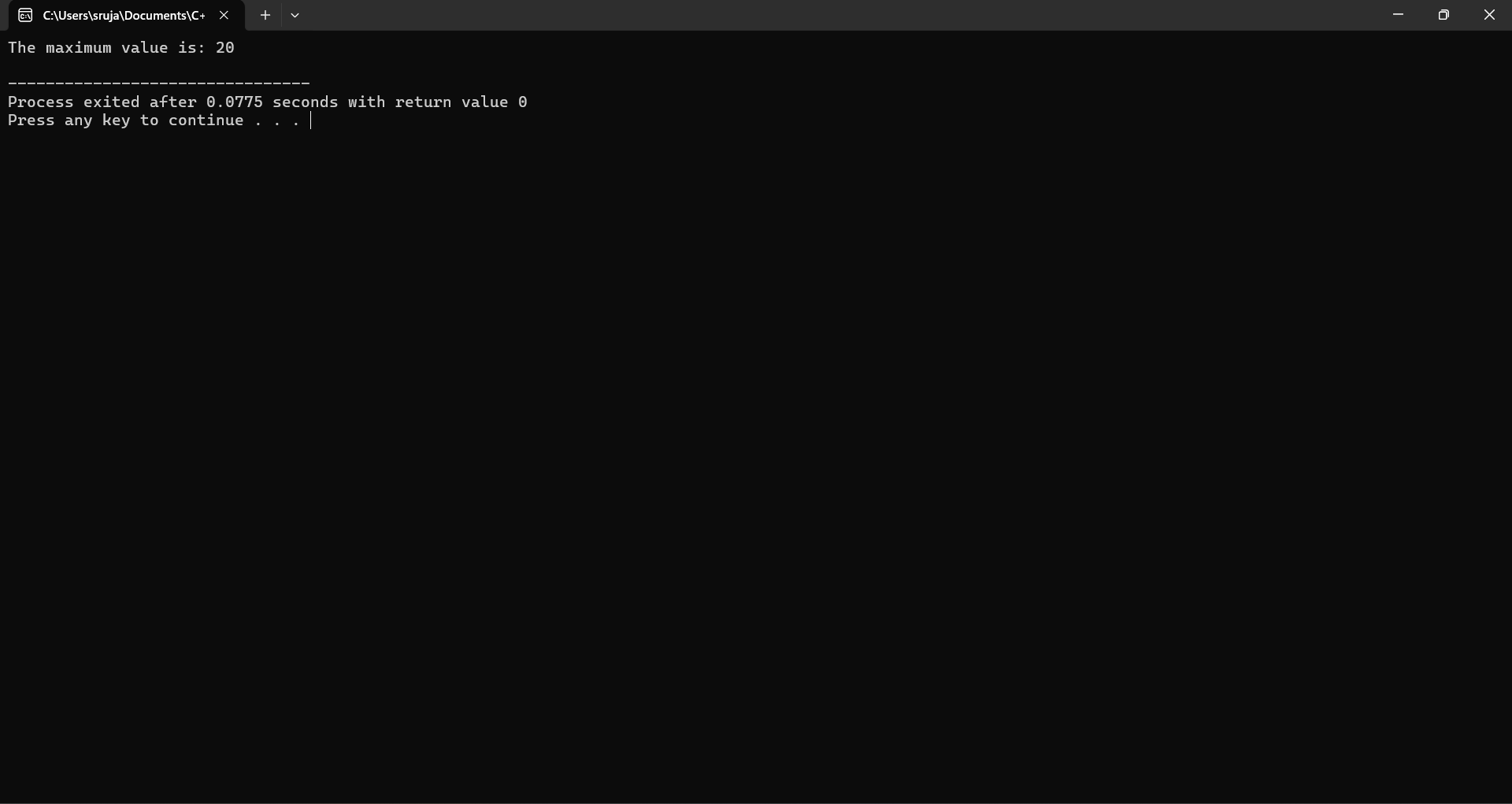
ClassB objB(20);

cout << "The maximum value is: " << max(objA, objB) << endl;

return 0;

}

**OUTPUT:**

****

**5.** Implement a banking system using C++ classes. Create classes for customers, accounts, and transactions. Apply encapsulation to protect sensitive information, and demonstrate the use of friend functions for access control. Discuss how encapsulation enhances the security and maintainability of the system.

**PROGRAM:**

#include <iostream>

#include <string>

#include <vector>

using namespace std;

class Customer;

class Account;

class Transaction;

class Customer {

private:

string name;

string address;

string phone;

public:

Customer(const string &n, const string &a, const string &p) : name(n), address(a), phone(p) {}

void displayCustomerInfo() const {

cout << "Customer Name: " << name << "\nAddress: " << address << "\nPhone: " << phone << endl;

}

// Friend function to allow access to private members

friend class Account;

};

class Account {

private:

int accountNumber;

double balance;

Customer \*customer;

public:

Account(int accNum, Customer \*cust) : accountNumber(accNum), balance(0.0), customer(cust) {}

void deposit(double amount) {

balance += amount;

cout << "Deposited $" << amount << ". New balance: $" << balance << endl;

}

void withdraw(double amount) {

if (amount > balance) {

cout << "Insufficient funds. Withdrawal failed." << endl;

} else {

balance -= amount;

cout << "Withdrew $" << amount << ". New balance: $" << balance << endl;

}

}

void displayAccountInfo() const {

cout << "Account Number: " << accountNumber << "\nBalance: $" << balance << endl;

customer->displayCustomerInfo();

}

// Friend function to allow access to private members

friend class Transaction;

};

class Transaction {

public:

static void transfer(Account &from, Account &to, double amount) {

if (amount > from.balance) {

cout << "Transfer failed. Insufficient funds in source account." << endl;

} else {

from.balance -= amount;

to.balance += amount;

cout << "Transferred $" << amount << " from Account " << from.accountNumber << " to Account " << to.accountNumber << endl;

}

}

};

int main() {

Customer cust1("John Doe", "123 Elm St", "555-1234");

Customer cust2("Jane Smith", "456 Oak St", "555-5678");

Account acc1(1001, &cust1);

Account acc2(1002, &cust2);

acc1.deposit(500);

acc2.deposit(300);

acc1.withdraw(100);

acc1.displayAccountInfo();

acc2.displayAccountInfo();

Transaction::transfer(acc1, acc2, 200);

acc1.displayAccountInfo();

acc2.displayAccountInfo();

return 0;

}

**OUTPUT:**

