**Assignment 3**

(1). Define a class MyClass with default constructor that initializes the member

variable x to 0. In main function, an object of MyClass is created and the value of x is

printed .Call the Myclass and passes different values in the Parameter and print

the values.

**Project:**

#include <iostream>

class MyClass {

private:

int x;

public:

MyClass() : x(0) {}

MyClass(int value) : x(value) {}

int getX() {

return x;

}

};

int main() {

MyClass obj1;

std::cout << "Value of x (default constructor): " << obj1.getX() << std::endl;

MyClass obj2(10);

std::cout << "Value of x (parameterized constructor): " << obj2.getX() << std::endl;

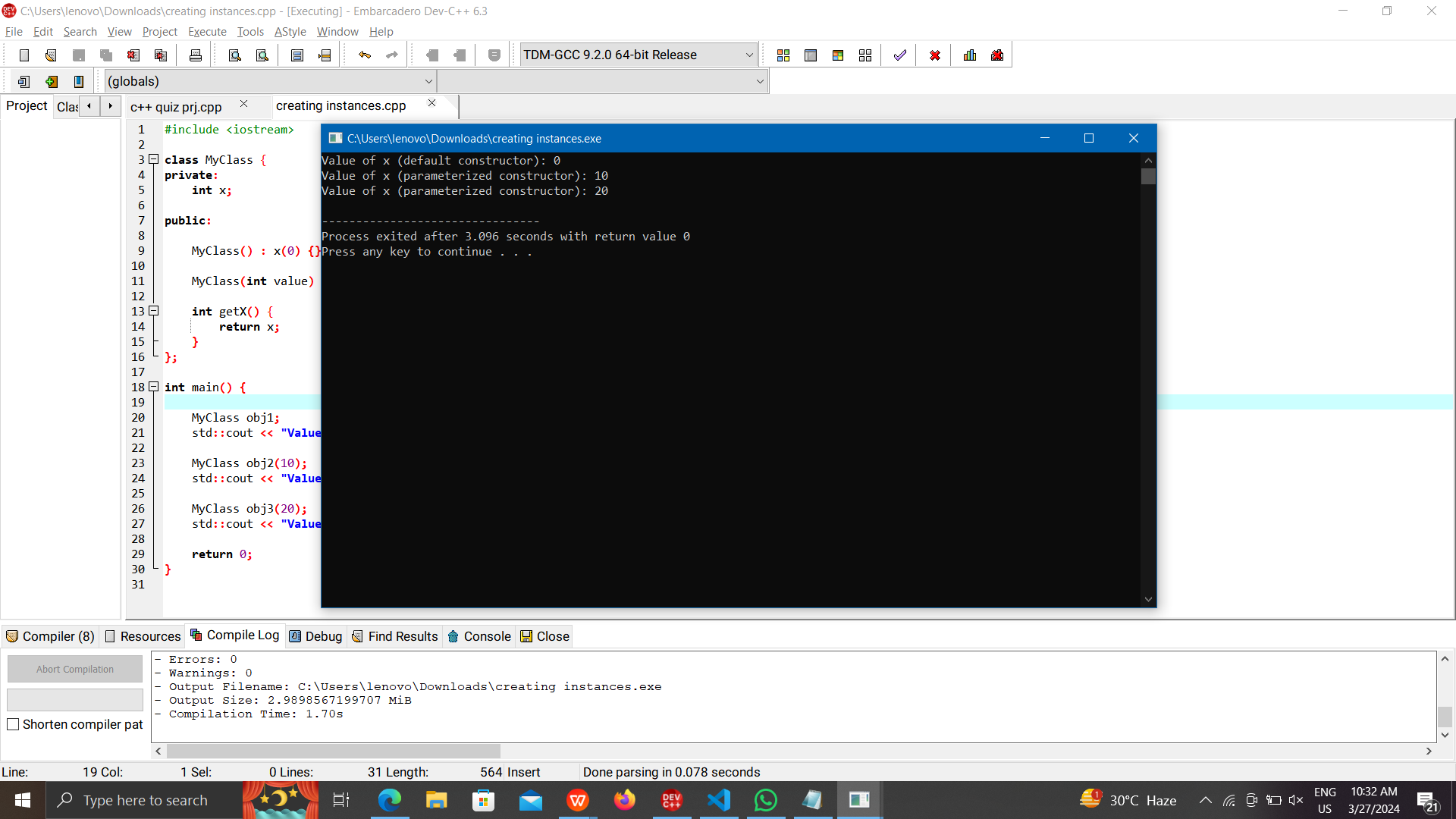
MyClass obj3(20);

std::cout << "Value of x (parameterized constructor): " << obj3.getX() << std::endl;

return 0;

}

**Output:**



(2). Develop a program how to use

parameterized constructor to initialize the

data members of class with user defined

values for Library Management Process.

i) Getting the Book Details(Book ID,

Name) in Stack

ii) User Details(Reg.No,Name) in Stack

iii) Check the Availability of book in Stack

iv) Distribute the Book for the User

**Program:** #include <iostream>

#include <stack>

#include <string>

class Book {

private:

int bookID;

std::string bookName;

public:

Book(int id, const std::string& name) : bookID(id), bookName(name) {}

int getBookID() {

return bookID;

}

std::string getBookName() {

return bookName;

}

};

class User {

private:

int regNo;

std::string userName;

public:

User(int reg, const std::string& name) : regNo(reg), userName(name) {}

int getRegNo() {

return regNo;

}

std::string getUserName() {

return userName;

}

};

int main() {

std::stack<Book> bookStack;

std::stack<User> userStack;

Book book1(101, "Introduction to C++");

Book book2(102, "Data Structures and Algorithms");

bookStack.push(book1);

bookStack.push(book2);

User user1(1001, "Alice");

User user2(1002, "Bob");

userStack.push(user1);

userStack.push(user2);

int bookIDToCheck = 101; // Book ID to check

bool bookAvailable = false;

std::stack<Book> tempStack = bookStack;

while (!tempStack.empty()) {

if (tempStack.top().getBookID() == bookIDToCheck) {

bookAvailable = true;

break;

}

tempStack.pop();

}

if (bookAvailable) {

std::cout << "Book with ID " << bookIDToCheck << " is available." << std::endl;

} else {

std::cout << "Book with ID " << bookIDToCheck << " is not available." << std::endl;

}

if (!bookStack.empty() && !userStack.empty()) {

Book distributedBook = bookStack.top();

User receivingUser = userStack.top();

std::cout << "Distributing Book: " << distributedBook.getBookName() << " to User: " << receivingUser.getUserName() << std::endl;

bookStack.pop();

userStack.pop();

} else {

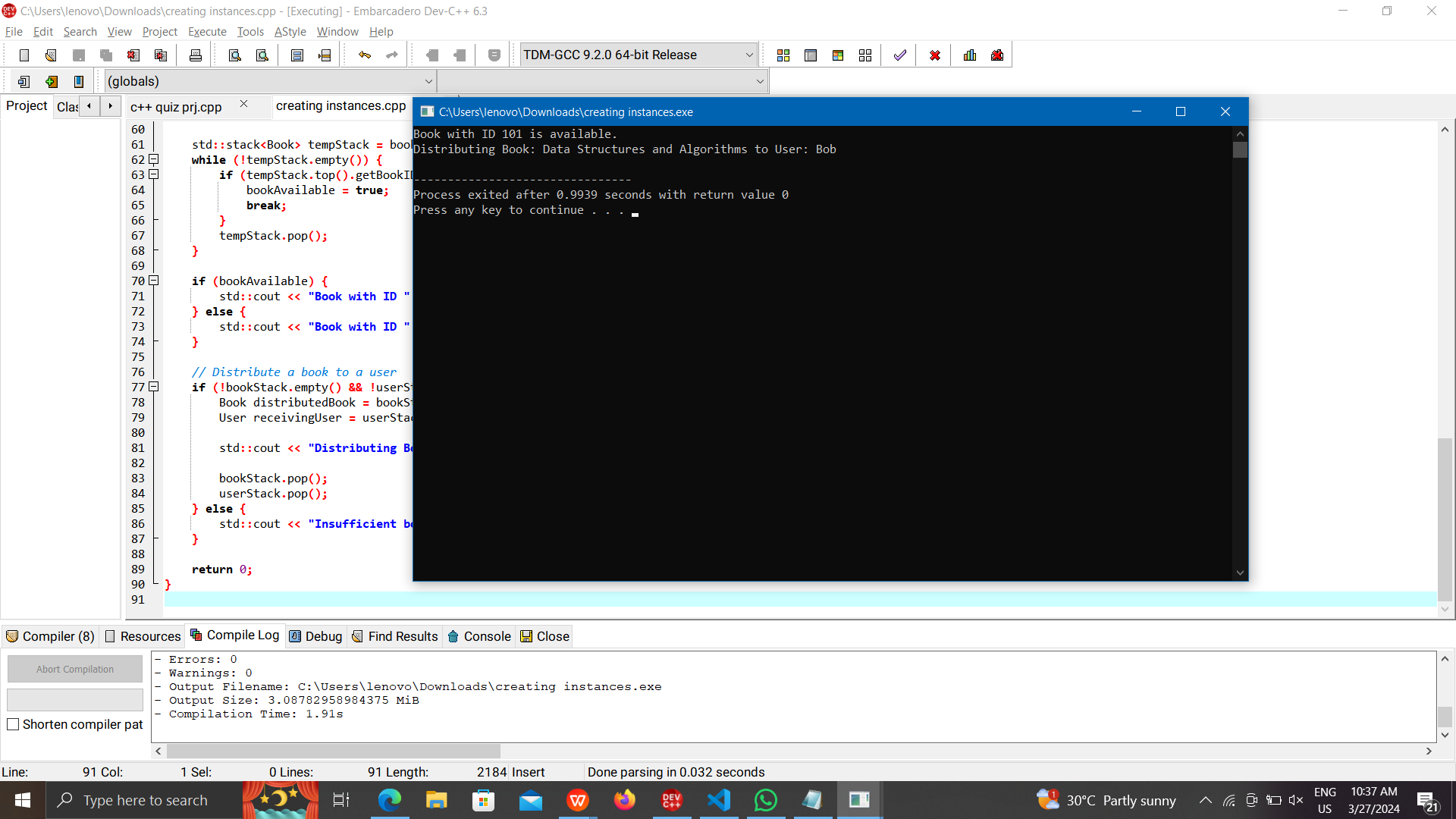
std::cout << "Insufficient books or users for distribution." << std::endl;

}

return 0;

}

**Output:**



(3). Write a program that demonstrates the use

of a copy constructor in C++ for Vehicle

Entry in College Campus.

i) Getting the Vehicle Details (ID, Vehicle

Type) in Queue.

ii) Assign the ID to the Users.

iii) Check the Availability of Place in

Queue.

iv) Allotment of the space in Queue

**Program:** #include <iostream>

#include <queue>

#include <string>

class Vehicle {

private:

int vehicleID;

std::string vehicleType;

public:

Vehicle(int id, const std::string& type) : vehicleID(id), vehicleType(type) {}

Vehicle(const Vehicle& other) : vehicleID(other.vehicleID), vehicleType(other.vehicleType) {

std::cout << "Copy constructor called." << std::endl;

}

int getVehicleID() {

return vehicleID;

}

std::string getVehicleType() {

return vehicleType;

}

};

int main() {

std::queue<Vehicle> vehicleQueue;

Vehicle vehicle1(101, "Car");

Vehicle vehicle2(102, "Motorcycle");

vehicleQueue.push(vehicle1);

vehicleQueue.push(vehicle2);

int userID = 1001; // User ID

Vehicle userVehicle = vehicleQueue.front();

vehicleQueue.pop();

if (!vehicleQueue.empty()) {

std::cout << "Space available in the queue for more vehicles." << std::endl;

} else {

std::cout << "No space available in the queue for more vehicles." << std::endl;

}

Vehicle newVehicle(103, "SUV");

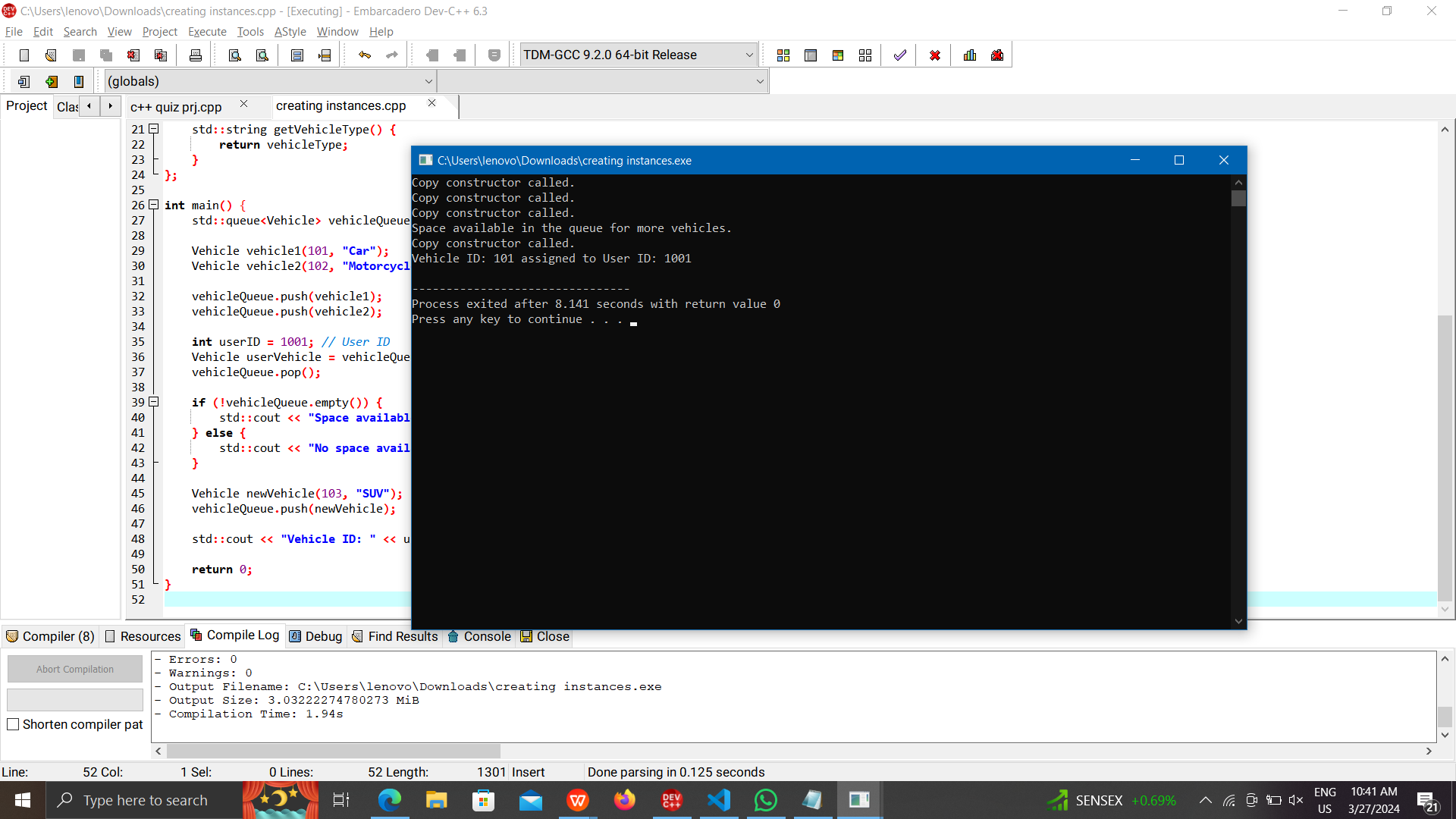
vehicleQueue.push(newVehicle);

std::cout << "Vehicle ID: " << userVehicle.getVehicleID() << " assigned to User ID: " << userID << std::endl;

return 0;

}

**Output:**



(4). Write a C++ Program to concatenate two string using unary operators overloading with sample input and output.

**Program:** #include <iostream>

#include <string>

class Concatenator {

private:

std::string concatenatedString;

public:

Concatenator() : concatenatedString("") {}

Concatenator operator+(const Concatenator& other) {

Concatenator result;

result.concatenatedString = concatenatedString + other.concatenatedString;

return result;

}

void display() {

std::cout << "Concatenated String: " << concatenatedString << std::endl;

}

void setString(const std::string& str) {

concatenatedString = str;

}

};

int main() {

Concatenator str1, str2, result;

str1.setString("Trespassers ");

str2.setString("will be prosecuted");

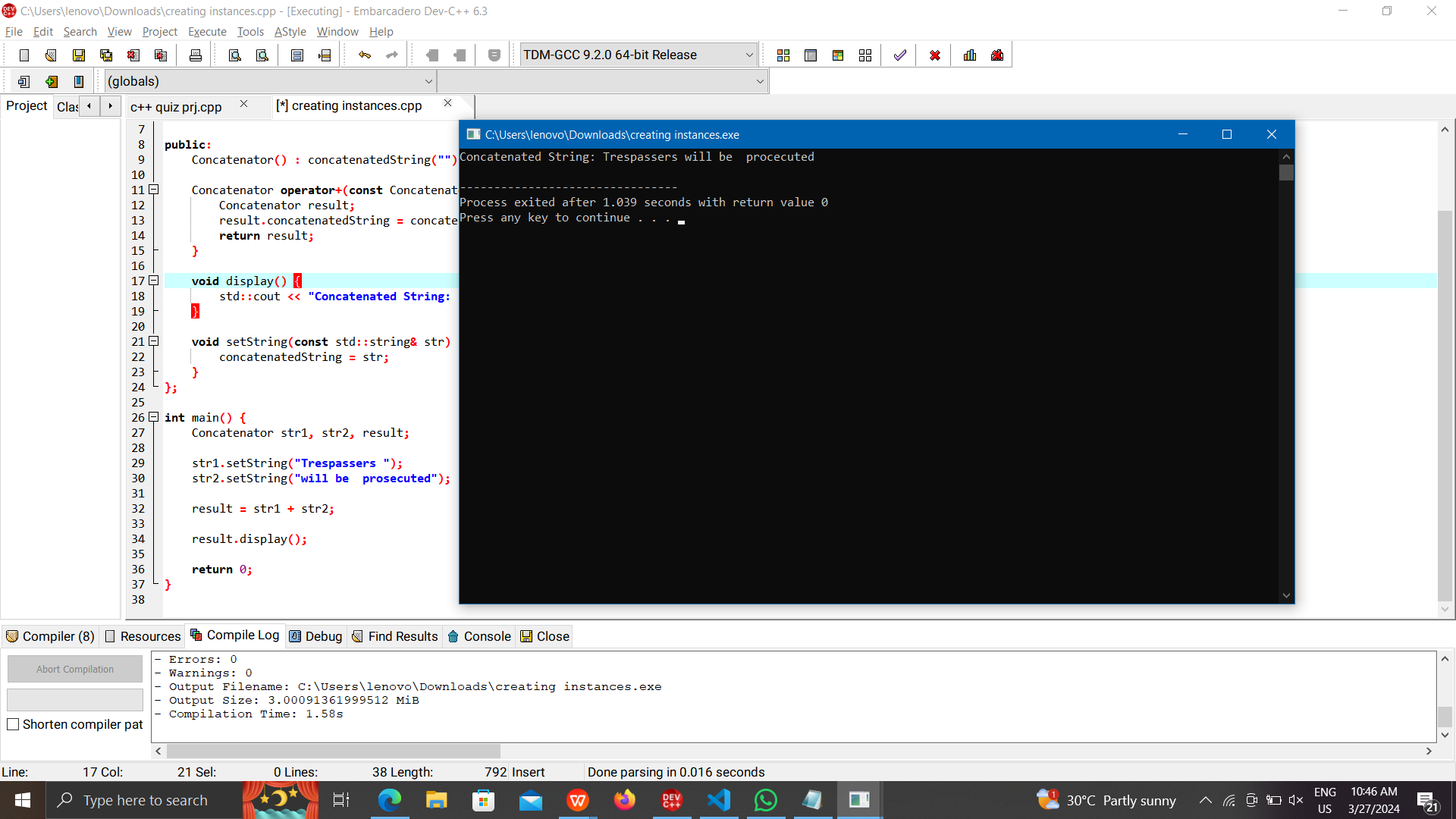
result = str1 + str2;

result.display();

return 0;

}

**Output:**



(5). Write a C++ program to demonstrate

operator overloading using dot operator for

i) Perfect Number checking

ii) Armstrong Number Checking

**Program:** #include <iostream>

#include <cmath>

class NumberChecker {

private:

int number;

public:

NumberChecker(int num) : number(num) {}

bool operator!() {

return isPerfectNumber();

}

bool operator~() {

return isArmstrongNumber();

}

bool isPerfectNumber() {

int sum = 0;

for (int i = 1; i <= number / 2; ++i) {

if (number % i == 0) {

sum += i;

}

}

return sum == number;

}

bool isArmstrongNumber() {

int originalNumber = number;

int numDigits = 0;

int sum = 0;

while (originalNumber != 0) {

originalNumber /= 10;

++numDigits;

}

originalNumber = number;

while (originalNumber != 0) {

int digit = originalNumber % 10;

sum += pow(digit, numDigits);

originalNumber /= 10;

}

return sum == number;

}

};

int main() {

NumberChecker num1(6);

NumberChecker num2(371);

if (!num1) {

std::cout << "Number 6 is a Perfect Number." << std::endl;

} else {

std::cout << "Number 6 is not a Perfect Number." << std::endl;

}

if (~num2) {

std::cout << "Number 371 is an Armstrong Number." << std::endl;

} else {

std::cout << "Number 371 is not an Armstrong Number." << std::endl;

}

return 0;

}

**Output:**

