**ASSIGNMENT 3**

CO1 : **Implement the concept of Constructors and operator overloading in programs.**

**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.**

**CODE**

#include <iostream>

class MyClass {

public:

MyClass() : x(0) {}

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

int x;

};

int main() {

MyClass obj1;

std::cout << "Value of x for obj1: " << obj1.x << std::endl;

MyClass obj2(5);

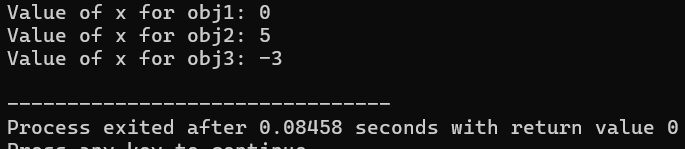
std::cout << "Value of x for obj2: " << obj2.x << std::endl;

MyClass obj3(-3);

std::cout << "Value of x for obj3: " << obj3.x << std::endl;

return 0;

}



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**

**CODE**

#include <iostream>

#include <stack>

#include <string>

using namespace std;

class Book {

public:

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

int getBookID() const { return bookID; }

const string& getBookName() const { return bookName; }

private:

int bookID;

string bookName;

};

class User {

public:

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

int getRegistrationNo() const { return registrationNo; }

const string& getUserName() const { return userName; }

private:

int registrationNo;

string userName;

};

class Library {

public:

void addBook(const Book& book) {

bookStack.push(book);

cout << "Book added to the stack: " << book.getBookName() << endl;

}

void addUser(const User& user) {

userStack.push(user);

cout << "User added to the stack: " << user.getUserName() << endl;

}

bool checkBookAvailability(int bookID) {

stack<Book> tempStack = bookStack;

while (!tempStack.empty()) {

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

cout << "Book " << tempStack.top().getBookName() << " is available.\n";

return true;

}

tempStack.pop();

}

cout << "Book with ID " << bookID << " is not available.\n";

return false;

}

// Function to distribute book to user

void distributeBook(int bookID, const User& user) {

if (checkBookAvailability(bookID)) {

cout << "Distributing book " << bookID << " to user " << user.getUserName() << endl;

bookStack.pop();

} else {

cout << "Book distribution failed.\n";

}

}

private:

stack<Book> bookStack;

stack<User> userStack;

};

int main() {

Book book1(101, "Book A");

Book book2(102, "Book B");

User user1(1001, "User X");

User user2(1002, "User Y");

// Create library object

Library library;

library.addBook(book1);

library.addBook(book2);

library.addUser(user1);

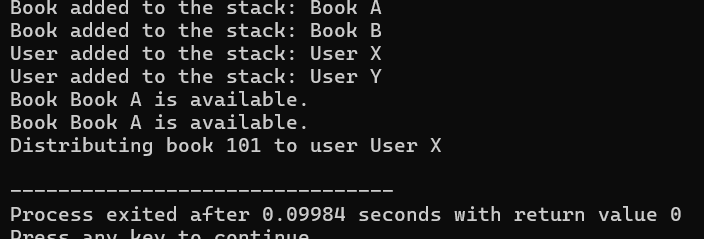
library.addUser(user2);

library.checkBookAvailability(101);

library.distributeBook(101, user1);

return 0;

}



**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 CO3 K3 10 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**

**CODE**

#include <iostream>

#include <queue>

#include <string>

using namespace std;

class Vehicle {

public:

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

int getVehicleID() const { return vehicleID; }

const string& getVehicleType() const { return vehicleType; }

private:

int vehicleID;

string vehicleType;

};

class CollegeCampus {

public:

// Default constructor

CollegeCampus() {}

// Copy constructor

CollegeCampus(const CollegeCampus& other) {

// Copy the queue contents

vehicleQueue = other.vehicleQueue;

}

void addVehicle(const Vehicle& vehicle) {

vehicleQueue.push(vehicle);

cout << "Vehicle added to the queue: ID " << vehicle.getVehicleID() << ", Type " << vehicle.getVehicleType() << endl;

}

void assignIDToUser(int userID) {

if (!vehicleQueue.empty()) {

cout << "User with ID " << userID << " assigned to vehicle ID " << vehicleQueue.front().getVehicleID() << endl;

vehicleQueue.pop();

} else {

cout << "No vehicles available to assign." << endl;

}

}

bool checkQueueSpace() const {

return !vehicleQueue.empty();

}

void allotSpaceInQueue(const Vehicle& vehicle) {

vehicleQueue.push(vehicle);

cout << "Space allotted in the queue for vehicle ID " << vehicle.getVehicleID() << endl;

}

private:

queue<Vehicle> vehicleQueue;

};

int main() {

Vehicle vehicle1(1001, "Car");

Vehicle vehicle2(1002, "Motorcycle");

CollegeCampus campus;

campus.addVehicle(vehicle1);

campus.addVehicle(vehicle2);

if (campus.checkQueueSpace()) {

campus.assignIDToUser(2001);

campus.assignIDToUser(2002);

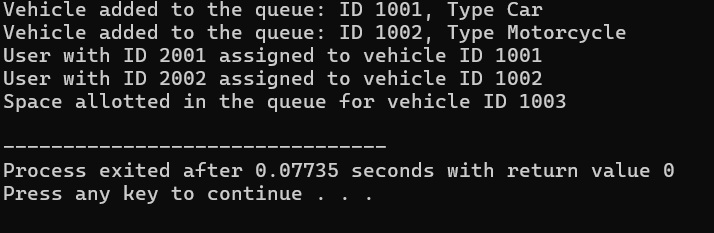
}

Vehicle newVehicle(1003, "Scooter");

campus.allotSpaceInQueue(newVehicle);

return 0;

}



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

**CODE**

#include <iostream>

#include <string>

using namespace std;

class Concatenator {

public:

Concatenator(const string& str) : concatString(str) {}

Concatenator operator+() const {

return \*this;

}

Concatenator operator+(const Concatenator& other) const {

Concatenator result = \*this;

result.concatString += other.concatString;

return result;

}

const string& getConcatString() const {

return concatString;

}

private:

string concatString;

};

int main() {

Concatenator str1("Hello, ");

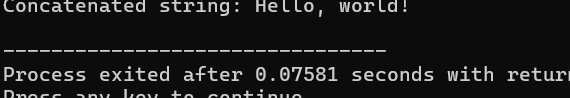
Concatenator str2("world!");

Concatenator result = +str1 + str2;

cout << "Concatenated string: " << result.getConcatString() << endl;

return 0;

}



**5.** Write a C++ program to demonstrate operator overloading using dot operator for

1. Perfect Number checking

Armstrong Number Checking

**CODE**

#include <iostream>

#include <cmath>

using namespace std;

class Number {

private:

int num;

public:

// Constructor

Number(int n) : num(n) {}

// Overloading dot operator for perfect number checking

bool operator.() {

int sum = 0;

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

if (num % i == 0) {

sum += i;

}

}

return sum == num;

}

// Overloading dot operator for Armstrong number checking

bool operator..() {

int originalNum = num;

int digits = 0;

int sum = 0;

// Count the number of digits

while (originalNum != 0) {

originalNum /= 10;

++digits;

}

originalNum = num;

// Compute the sum of nth power of individual digits

while (originalNum != 0) {

int digit = originalNum % 10;

sum += pow(digit, digits);

originalNum /= 10;

}

return sum == num;

}

};

int main() {

int num;

cout << "Enter a number: ";

cin >> num;

Number n(num);

if (n.) {

cout << num << " is a perfect number." << endl;

} else {

cout << num << " is not a perfect number." << endl;

}

if (n..()) {

cout << num << " is an Armstrong number." << endl;

} else {

cout << num << " is not an Armstrong number." << endl;

}

return 0;

}

