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.**

#include <iostream>

using namespace std;

class MyClass {

private:

int x;

public:

MyClass() : x(0) {}

int getX() const {

return x;

}

};

int main() {

MyClass obj1;

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

MyClass obj2(10);

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

MyClass obj3(20);

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

return 0;

}

OUTPUT:

Value of x using default constructor: 0

Value of x using parameterized constructor: 10

Value of x using parameterized constructor: 20

**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**

#include <iostream>

#include <string>

#include <stack>

using namespace std;

class Book {

private:

int bookId;

string bookName;

public:

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

int getBookId() const {

return bookId;

}

const string& getBookName() const {

return bookName;

}

};

class User {

private:

int regNo;

string userName;

public:

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

int getRegNo() const {

return regNo;

const string& getUserName() const {

return userName;

}

};

int main() {

stack<Book> bookStack;

bookStack.push(Book(101, "Book1"));

bookStack.push(Book(102, "Book2"));

bookStack.push(Book(103, "Book3"));

userStack.push(User(1001, "User1"));

userStack.push(User(1002, "User2"));

userStack.push(User(1003, "User3"));

int bookIdToCheck = 102;

bool foundBook = false;

stack<Book> tempBookStack;

while (!bookStack.empty()) {

if (bookStack.top().getBookId() == bookIdToCheck) {

foundBook = true;

break;

}

tempBookStack.push(bookStack.top());

bookStack.pop();

}

while (!tempBookStack.empty()) {

bookStack.push(tempBookStack.top());

tempBookStack.pop();

}

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

Book distributedBook = bookStack.top();

User recipient = userStack.top();

cout << "Book ID: " << distributedBook.getBookId() << ", Book Name: " << distributedBook.getBookName()

<< " distributed to User ID: " << recipient.getRegNo() << ", User Name: " << recipient.getUserName() << endl;

bookStack.pop();

userStack.pop();

} else {

cout << "Book not available or no users in the system." << endl;

}

return 0;

}

OUTPUT:

Book ID: 103

Book Name: Book3

distributed to User ID: 1003

User Name: User3

**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.**

#include <iostream>

#include <string>

#include <queue>

using namespace std;

class Vehicle {

private:

int vehicleId;

string vehicleType;

public:

Vehicle() : vehicleId(0), vehicleType("Car") {}

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

Vehicle(const Vehicle& other) {

vehicleId = other.vehicleId;

vehicleType = other.vehicleType;

}

int getVehicleId() const {

return vehicleId;

}

const string& getVehicleType() const {

return vehicleType;

}

};

class User {

private:

int userId;

Vehicle vehicle;

public:

User(int id, const Vehicle& v) : userId(id), vehicle(v) {}

int getUserId() const {

return userId;

}

const Vehicle& getVehicle() const {

return vehicle;

}

};

int main() {

queue<Vehicle> vehicleQueue;

vehicleQueue.push(Vehicle(1, "Car"));

vehicleQueue.push(Vehicle(2, "Bike"));

vehicleQueue.push(Vehicle(3, "Scooter"));

if (!vehicleQueue.empty()) {

Vehicle allottedVehicle = vehicleQueue.front();

vehicleQueue.pop();

User user1(101, allottedVehicle);

cout << "User ID: " << user1.getUserId() << endl;

cout << "Vehicle ID: " << user1.getVehicle().getVehicleId() << ", Vehicle Type: " << user1.getVehicle().getVehicleType() << endl;

} else {

cout << "No space available in the queue." << endl;

}

return 0;

}

OUTPUT:

User ID: 101

Vehicle ID: 1

Vehicle Type: Car

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

#include <iostream>

#include <cstring>

using namespace std;

class AddString {

public:

char s1[25], s2[25];

AddString(char str1[], char str2[]) {

strcpy(this->s1, str1);

strcpy(this->s2, str2);

}

void operator+() {

cout << "\nConcatenation: " << strcat(s1, s2);

}

};

int main() {

char str1[] = "Geeks";

char str2[] = "ForGeeks";

AddString a1(str1, str2);

+a1;

return 0;

}

OUTPUT:

Concatenation: GeeksForGeeks

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

**i) Perfect Number checking**

**ii) Armstrong Number Checking**

#include <iostream>

using namespace std;

class NumberChecker {

private:

int num;

public:

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

// Check if the number is perfect

bool isPerfect() {

int sum = 0;

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

if (num % i == 0) {

sum += i;

}

}

return sum == num;

}

// Check if the number is Armstrong

bool isArmstrong() {

int originalNum = num;

int result = 0;

while (originalNum != 0) {

int digit = originalNum % 10;

result += digit \* digit \* digit;

originalNum /= 10;

}

return result == num;

}

};

int main() {

int inputNum;

cout << "Enter a positive integer: ";

cin >> inputNum;

NumberChecker numChecker(inputNum);

if (numChecker.isPerfect()) {

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

} else {

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

}

if (numChecker.isArmstrong()) {

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

} else {

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

}

return 0;

}

OUTPUT:  
Enter a positive integer: 28

28 is a perfect number.

28 is not an Armstrong number.