Assignment No: 2

Title: Student Database using constructor, friend class, this pointer, inline code and dynamic memory allocation.

Problem Statement: Develop an object oriented program in C++ to create a database of student containing the following information: Roll No. Name, Date of Birth, Blood group, Class, Div, Contact address, telephone number, driving license no. etc. Construct the database with suitable member functions for initializing and destroying the data using friend class, this pointer, inline code and dynamic memory allocation operators-new and delete as well as exception handling.

#include <iostream>

#include <string>

#include <stdexcept> // For exception handling

using namespace std;

class Student; // Forward declaration of Student class

// Friend class declaration

class FriendClass {

public:

void displayStudentDetails(const Student& student); // Function to access private members of Student class

};

class Student {

private:

string name;

int rollNumber;

string className;

char division;

string dob;

string bloodGroup;

string address;

string telephoneNumber;

string drivingLicenseNo;

static int StudentCount; // Static member to track total students

public:

// Default Constructor

Student() : name(""), rollNumber(0), className(""), division(' '), dob(""), bloodGroup(""), address(""), telephoneNumber(""), drivingLicenseNo("") {

StudentCount++;

}

// Parameterized Constructor

Student(string n, int roll, string c, char div, string dob, string bg, string addr, string tel, string dl)

: name(n), rollNumber(roll), className(c), division(div), dob(dob), bloodGroup(bg), address(addr), telephoneNumber(tel), drivingLicenseNo(dl) {

StudentCount++;

}

// Destructor

~Student() {

StudentCount--;

}

// Static function to get the total student count

static int getStudentCount() {

return StudentCount;

}

// Function to input student details

void setValues() {

try {

cout << "Enter Name: ";

cin.ignore();

getline(cin, name);

cout << "Enter Roll Number: ";

cin >> rollNumber;

cin.ignore();

cout << "Enter Class: ";

getline(cin, className);

cout << "Enter Division: ";

cin >> division;

cin.ignore();

cout << "Enter Date of Birth (DD/MM/YYYY): ";

getline(cin, dob);

cout << "Enter Blood Group: ";

getline(cin, bloodGroup);

cout << "Enter Address: ";

getline(cin, address);

cout << "Enter Telephone Number: ";

getline(cin, telephoneNumber);

cout << "Enter Driving License No: ";

getline(cin, drivingLicenseNo);

} catch (const exception& e) {

cout << "Error while entering data: " << e.what() << endl;

}

}

// Inline function to display student details

inline void display() const {

cout << "Name: " << name << endl;

cout << "Roll Number: " << rollNumber << endl;

cout << "Class: " << className << endl;

cout << "Division: " << division << endl;

cout << "Date of Birth: " << dob << endl;

cout << "Blood Group: " << bloodGroup << endl;

cout << "Address: " << address << endl;

cout << "Telephone Number: " << telephoneNumber << endl;

cout << "Driving License No: " << drivingLicenseNo << endl;

}

// Friend function to access private members

friend void FriendClass::displayStudentDetails(const Student& student);

};

int Student::StudentCount = 0; // Initialize static member

// Friend class definition

void FriendClass::displayStudentDetails(const Student& student) {

cout << "\nDisplaying details using FriendClass:\n";

cout << "Name: " << student.name << endl;

cout << "Roll Number: " << student.rollNumber << endl;

cout << "Class: " << student.className << endl;

cout << "Division: " << student.division << endl;

cout << "Date of Birth: " << student.dob << endl;

cout << "Blood Group: " << student.bloodGroup << endl;

cout << "Address: " << student.address << endl;

cout << "Telephone Number: " << student.telephoneNumber << endl;

cout << "Driving License No: " << student.drivingLicenseNo << endl;

}

int main() {

int no;

try {

cout << "Enter the number of Students: ";

cin >> no;

// Dynamic memory allocation with exception handling

Student\* students = new Student[no];

// Handle memory allocation failure

if (students == nullptr) {

throw runtime\_error("Memory allocation failed!");

}

// Input student details

for (int i = 0; i < no; i++) {

cout << "\nEntering details for Student " << (i + 1) << ":" << endl;

students[i].setValues();

}

// Display student details

cout << "\nStudent Details:\n";

for (int i = 0; i < no; i++) {

cout << "\nStudent " << (i + 1) << " Details:" << endl;

students[i].display();

}

cout << "\nTotal Number of Students: " << Student::getStudentCount() << endl;

// Display details using FriendClass

FriendClass friendClass;

for (int i = 0; i < no; i++) {

friendClass.displayStudentDetails(students[i]);

}

// Free dynamically allocated memory

delete[] students; // Free dynamically allocated memory

} catch (const bad\_alloc& e) {

cout << "Memory allocation error: " << e.what() << endl;

} catch (const exception& e) {

cout << "Error: " << e.what() << endl;

}

return 0;

}