```cpp
// student_db.cpp
#include <iostream>
#include <string>
#include <stdexcept>
using namespace std;

class Student {
private:
    string *name;          // dynamic to demonstrate new/delete
    int roll;
    string dob;
    static int studentCount;

public:
    // Default constructor
    Student(): name(new string("Unknown")), roll(0), dob("00-00-0000") {
        ++studentCount;
    }

    // Parameterized constructor
    Student(const string &n, int r, const string &d) : name(new string(n)), roll(r), dob(d) {
        ++studentCount;
    }

    // Copy constructor
    Student(const Student &s) : name(new string(*s.name)), roll(s.roll), dob(s.dob) {
        ++studentCount;
    }

    // Assignment operator (deep copy)
    Student& operator=(const Student &s) {
        if (this != &s) {
            delete name;
            name = new string(*s.name);
            roll = s.roll;
            dob = s.dob;
        }
        return *this;
    }

    // Destructor
    ~Student() {
        delete name;
        --studentCount;
    }

    // Static member function
    static int getStudentCount() {
        return studentCount;
    }

    // Set data (demonstrates use of this)
    void setData(const string &n, int r, const string &d) {
        if (r <= 0) throw invalid_argument("Roll number must be positive");
        *name = n;
        this->roll = r; // using this pointer
        dob = d;
    }

    // Friend to display (access private data)
    friend void display(const Student &s);
};

int Student::studentCount = 0;

void display(const Student &s) {
    cout << "Name: " << *s.name << ", Roll: " << s.roll << ", DOB: " << s.dob << "\n";
}

int main() {
    try {
        Student s1("Pranav", 101, "01-01-2000");
        Student s2; // default
        s2.setData("Aman", 102, "05-05-2000");

        Student s3 = s1; // copy constructor
        display(s1);
        display(s2);
        display(s3);

        cout << "Count: " << Student::getStudentCount() << "\n";

        // demonstrate assignment and exception
        Student *s4 = new Student();
        try {
            s4->setData("Faulty", -5, "10-10-2000"); // will throw
        } catch (const invalid_argument &e) {
            cout << "Exception caught while setting student: " << e.what() << "\n";
        }
        delete s4;

        cout << "Final Count: " << Student::getStudentCount() << "\n";
    } catch (const exception &e) {
        cout << "Unhandled exception: " << e.what() << "\n";
    }
    return 0;
}
```