PROJECT REPORT

Report of Data Structure Project: Student Management System

Hubei University OF technology

Data Structure

The Report for the Course Exercise of Data Structures

Topic: Student Management System

Group: ADT

Members: 1911521109,1911521111,1911521112.1911562107

Data: User’s Manual

School of Computer science

Acknowledgment

The is document was produces as a *Report to Final Project* of Course “Data Structure” taken by ***Xinyun Wu*** by collaborative work of the group members of “ADT”. ADT consists of group members *Akhmetzhanova Anel* (1911521111), *Mohadmmad Mustakim Hassan* (1911521112), *Sony Hossain Ismail* (1911521109), and *Rahman Noushad A N M Nurullah Masuder* (1911562107).

Contents

Abstract

The main aim for our project entitled as “Student Management System” is to develop a computerized system to show how it manages all the daily work of school. Student Management System project is a smart and effective way to store records of the student on the basis of school and admins of that university. This project has many features, which we will describe to you in the following parts.   Overall, project of ours solves problem of manual work in the schools and helps the its system to maintain the data structure efficiently.

Problem definition & Demand analysis

Introduction

Researches demonstrated that, in order to satisfy the information demand of modern sociality the schools transferred to automated service model. Today, technology has been helping to manage universities/schools efficiently. Solutions are widely used and the software got developed Student Management System that uses to supervise and monitor the record and all necessary requirement to handle a school's day-to-day actions. Admin is the overall controls all the record and builds up the data and maintains it. The system provides set of features: registration of the student's details, assigning the department based on their course and maintenance of the record.

Problem Statement:

System must be able to handle actions like: login features, adding students/teacher, adding courses, adding marks, view details, edit/modify the student details, delete student from the record, also system should be able to inquire, sort and update the information.

Problem Definition:

- The Student Management system implements databases to make the existing system more structured
- A school's database system is an infrastructure that allows users to view and edit details
- Student must provide the user name and the ID number, which is unique and confidential to each user.

Demand analysis

It need to accurately obtain the user requirements, thus to define system requirements.

The main activity is focused for the admin only who is like a chairman to all the organization and setup the data.

Objective:

- Our objective in this project is to demonstrate Student Management System where in users can access all the features told above.

Additionally, which will provide the following capabilities:

- Provide additional flexibility and convenience to the system users.
- Provide better reliability and security of the private information.

- Provide a more productive environment for the school staff member.

We will use the structure of Linked List. For now, we will focus on the following set of requirements.

Main requirements:

1) User's account details (ID card, password, status, name, gender, address, email, phone)
2) Department (department ID, course ID)
3) Marks
4) List

Oher requirements:

1) Features (insert, update, sort, delete)
2) User-Friendly as the system is interactive and can be easily operated.

Result:

- Student Management System gives access to the resources. A well-organized system will increase school's efficiency, save valuable administration time, lead to a better educational experience for students.

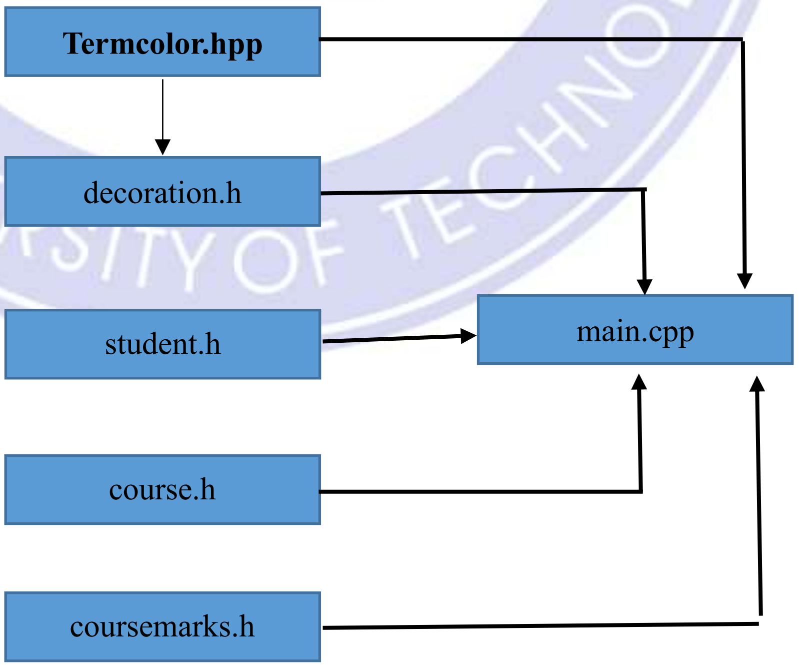
Once the planning and analysis of the project are completed, the design phase begins.

Design of the main framework

The program primarily has 5 modules. One primary entry point traditionally named "main, and 4 others based on different primary functionality included in the "include" folder as header files. The software also uses a third-party header only color library to color the UI.

- src (directory)
    1. main.cpp
    2. include (directory)
        I. student.h
        II. course.h
        III. coursemarks.h
        IV. decorations.h
        V. termcolor.hpp (third-party header-only library)

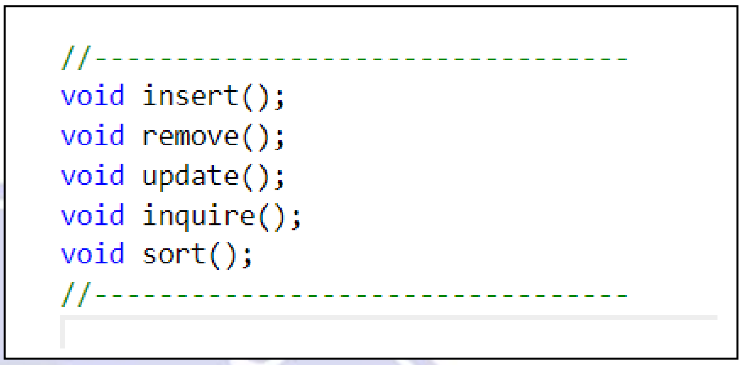
Their relationship is as follows:-



Detailed design

There few function:

- Insert
- Remove
- Update Info
- Inquire
- Sort Lists
- Exit

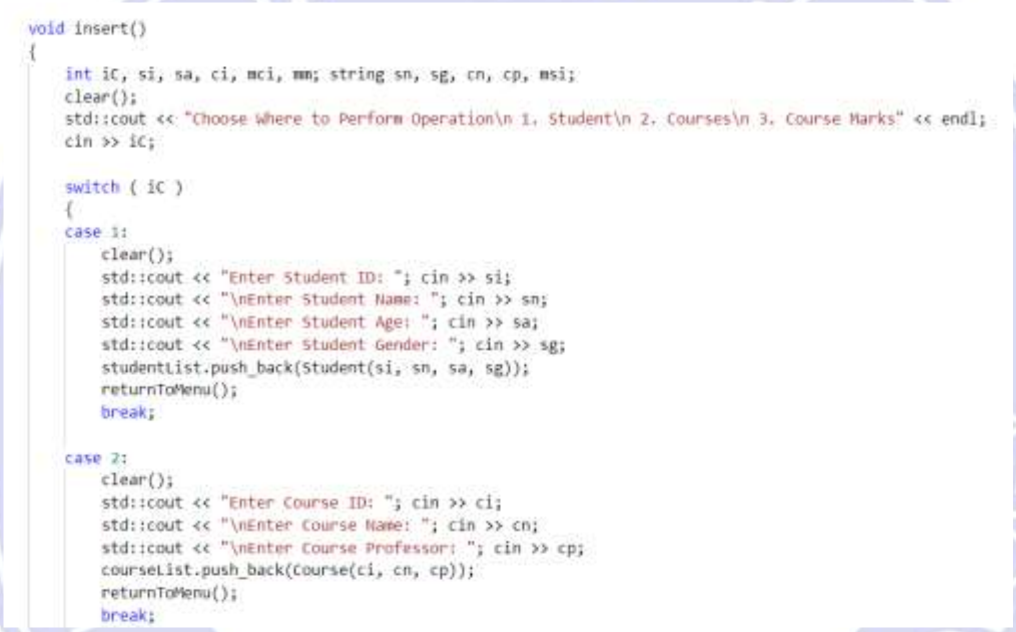


```
//--------------------------------
void insert();
void remove();
void update();
void inquire();
void sort();
//--------------------------------
```

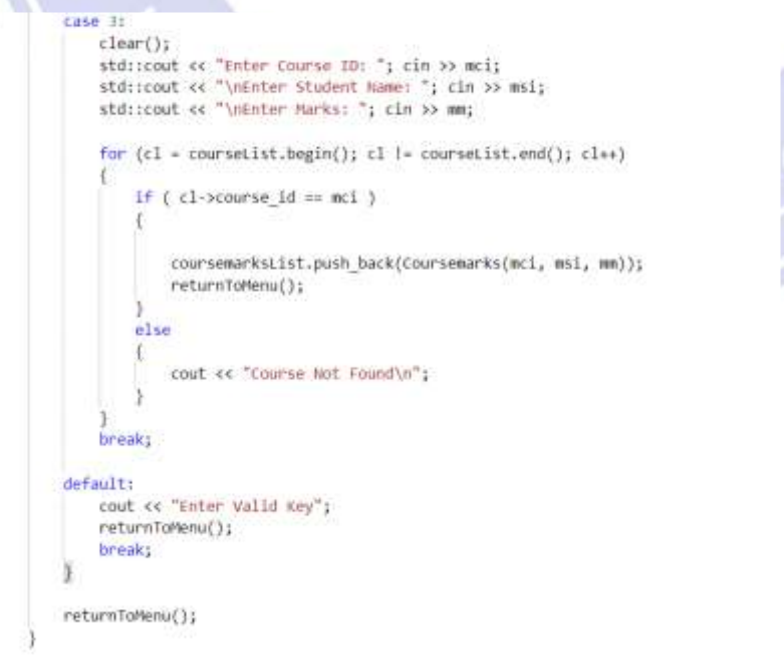
Each section has few function to operate:

Insert:

- Student
- Course
- Course Marks



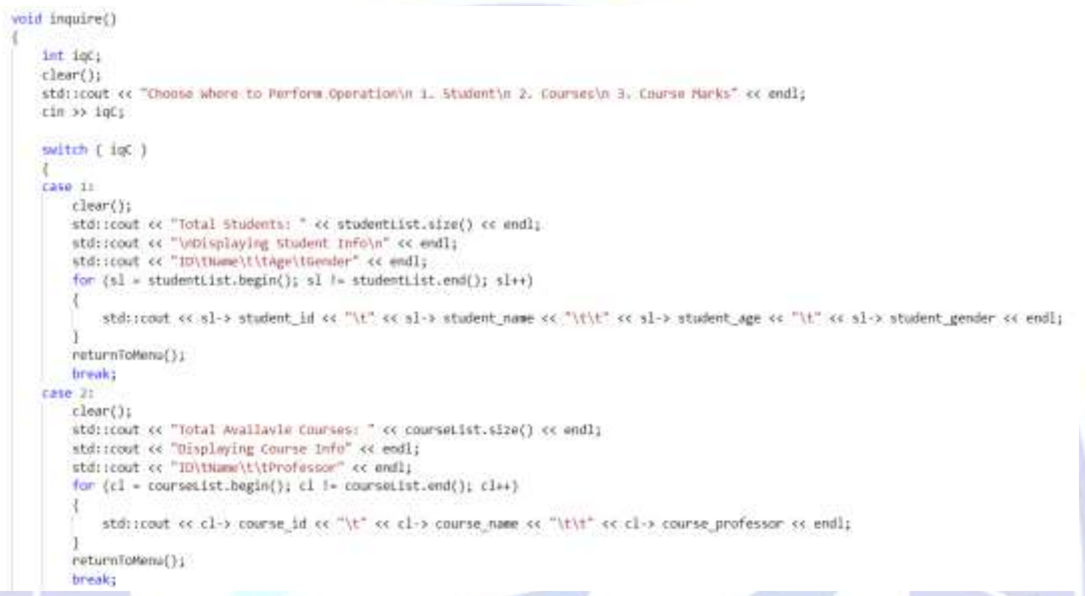
```
void insert()
{
    int iC, si, sa, ci, mci, mm; string sn, sg, cn, cp, msi;
    clear();
    std::cout << "Choose Where to Perform Operation\n 1. Student\n 2. Courses\n 3. Course Marks" << endl;
    cin >> iC;

    switch ( iC )
    {
    case 1:
        clear();
        std::cout << "Enter Student ID: "; cin >> si;
        std::cout << "\nEnter Student Name: "; cin >> sn;
        std::cout << "\nEnter Student Age: "; cin >> sa;
        std::cout << "\nEnter Student Gender: "; cin >> sg;
        studentList.push_back(Student(si, sn, sa, sg));
        returnToMenu();
        break;

    case 2:
        clear();
        std::cout << "Enter Course ID: "; cin >> ci;
        std::cout << "\nEnter Course Name: "; cin >> cn;
        std::cout << "\nEnter Course Professor: "; cin >> cp;
        courseList.push_back(Course(ci, cn, cp));
        returnToMenu();
        break;
```



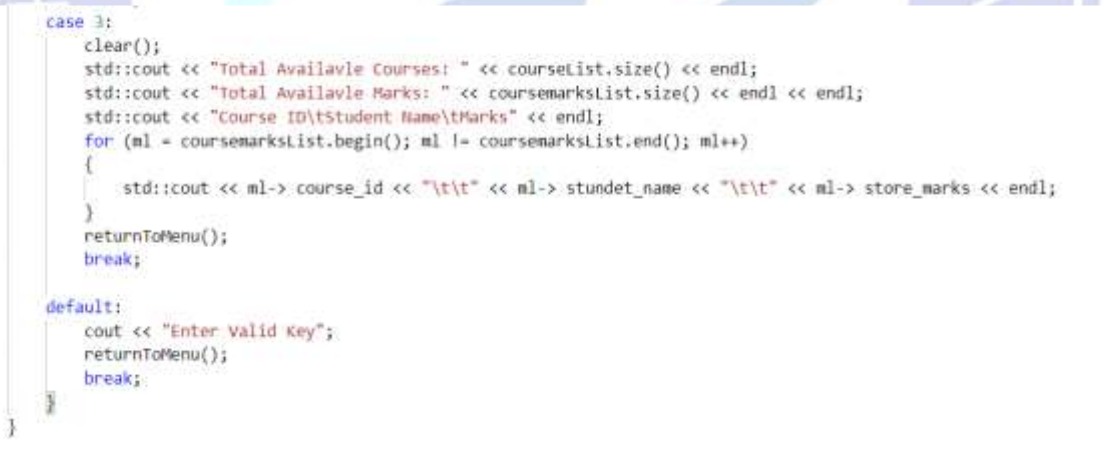
```
    case 3:
        clear();
        std::cout << "Enter Course ID: "; cin >> mci;
        std::cout << "\nEnter Student Name: "; cin >> msi;
        std::cout << "\nEnter Marks: "; cin >> mm;

        for (cl = courseList.begin(); cl != courseList.end(); cl++)
        {
            if ( cl->course_id == mci )
            {

                coursemarksList.push_back(Coursemarks(mci, msi, mm));
                returnToMenu();
            }
            else
            {
                cout << "Course Not Found\n";
            }
        }
        break;

    default:
        cout << "Enter Valid Key";
        returnToMenu();
        break;
    }

    returnToMenu();
}
```

Inquire:

- Student
- Course
- Course Marks



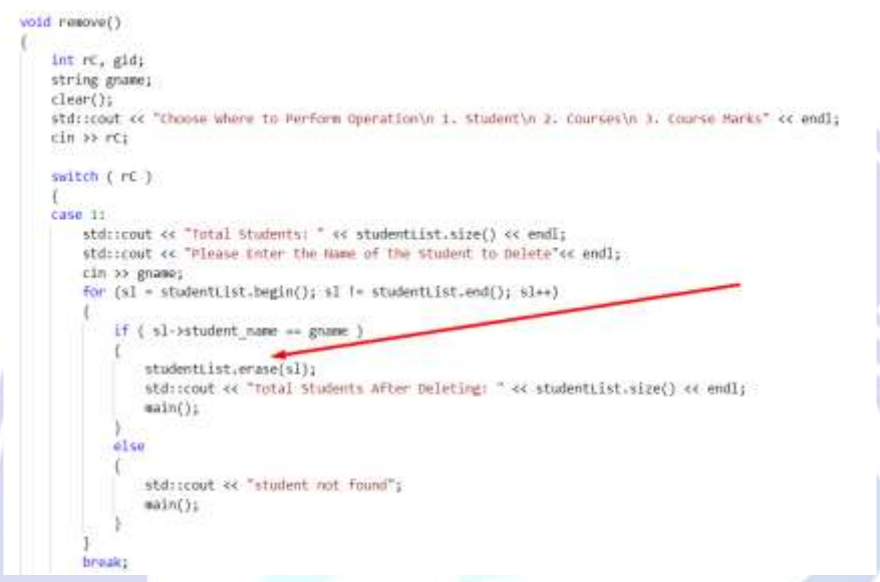
```cpp
void inquire()
{
    int iqC;
    clear();
    std::cout << "Choose where to Perform Operation\n 1. Student\n 2. Courses\n 3. Course Marks" << endl;
    cin >> iqC;

    switch ( iqC )
    {
    case 1:
        clear();
        std::cout << "Total Students: " << studentList.size() << endl;
        std::cout << "\nDisplaying Student Info\n" << endl;
        std::cout << "ID\tName\t\tAge\tGender" << endl;
        for (sl = studentList.begin(); sl != studentList.end(); sl++)
        {
            std::cout << sl-> student_id << "\t" << sl-> student_name << "\t\t" << sl-> student_age << "\t" << sl-> student_gender << endl;
        }
        returnToMenu();
        break;
    case 2:
        clear();
        std::cout << "Total Availavle Courses: " << courseList.size() << endl;
        std::cout << "Displaying Course Info" << endl;
        std::cout << "ID\tName\t\tProfessor" << endl;
        for (cl = courseList.begin(); cl != courseList.end(); cl++)
        {
            std::cout << cl-> course_id << "\t" << cl-> course_name << "\t\t" << cl-> course_professor << endl;
        }
        returnToMenu();
        break;
```



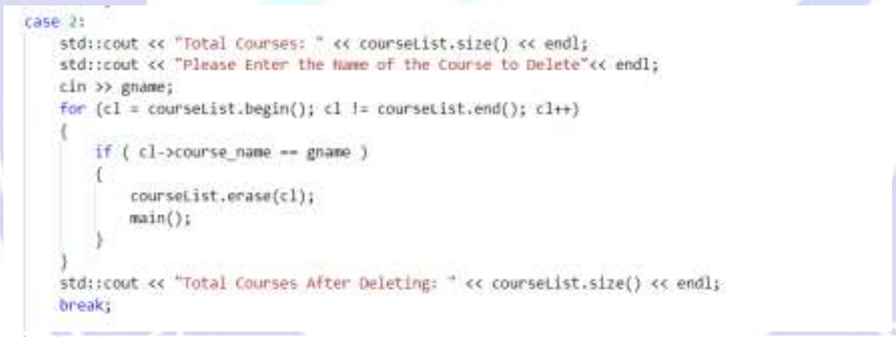
```cpp
    case 3:
        clear();
        std::cout << "Total Availavle Courses: " << courseList.size() << endl;
        std::cout << "Total Availavle Marks: " << coursemarksList.size() << endl << endl;
        std::cout << "Course ID\tStudent Name\tMarks" << endl;
        for (ml = coursemarksList.begin(); ml != coursemarksList.end(); ml++)
        {
            std::cout << ml-> course_id << "\t\t" << ml-> stundet_name << "\t\t" << ml-> store_marks << endl;
        }
        returnToMenu();
        break;

    default:
        cout << "Enter Valid Key";
        returnToMenu();
        break;
    }
}
```

Remove info:

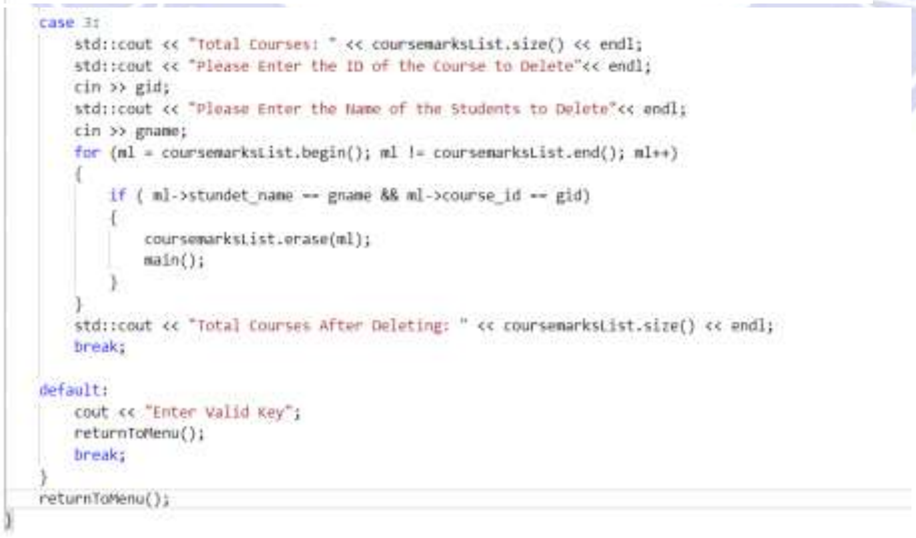
- Student
- Course
- Course Marks



```cpp
void remove()
{
    int rC, gid;
    string gname;
    clear();
    std::cout << "Choose Where to Perform Operation\n 1. Student\n 2. Courses\n 3. Course Marks" << endl;
    cin >> rC;

    switch ( rC )
    {
    case 1:
        std::cout << "Total Students: " << studentList.size() << endl;
        std::cout << "Please Enter the Name of the Student to Delete"<< endl;
        cin >> gname;
        for (sl = studentList.begin(); sl != studentList.end(); sl++)
        {
            if ( sl->student_name == gname )
            {
                studentList.erase(sl);
                std::cout << "Total Students After Deleting: " << studentList.size() << endl;
                main();
            }
            else
            {
                std::cout << "student not found";
                main();
            }
        }
        break;
```



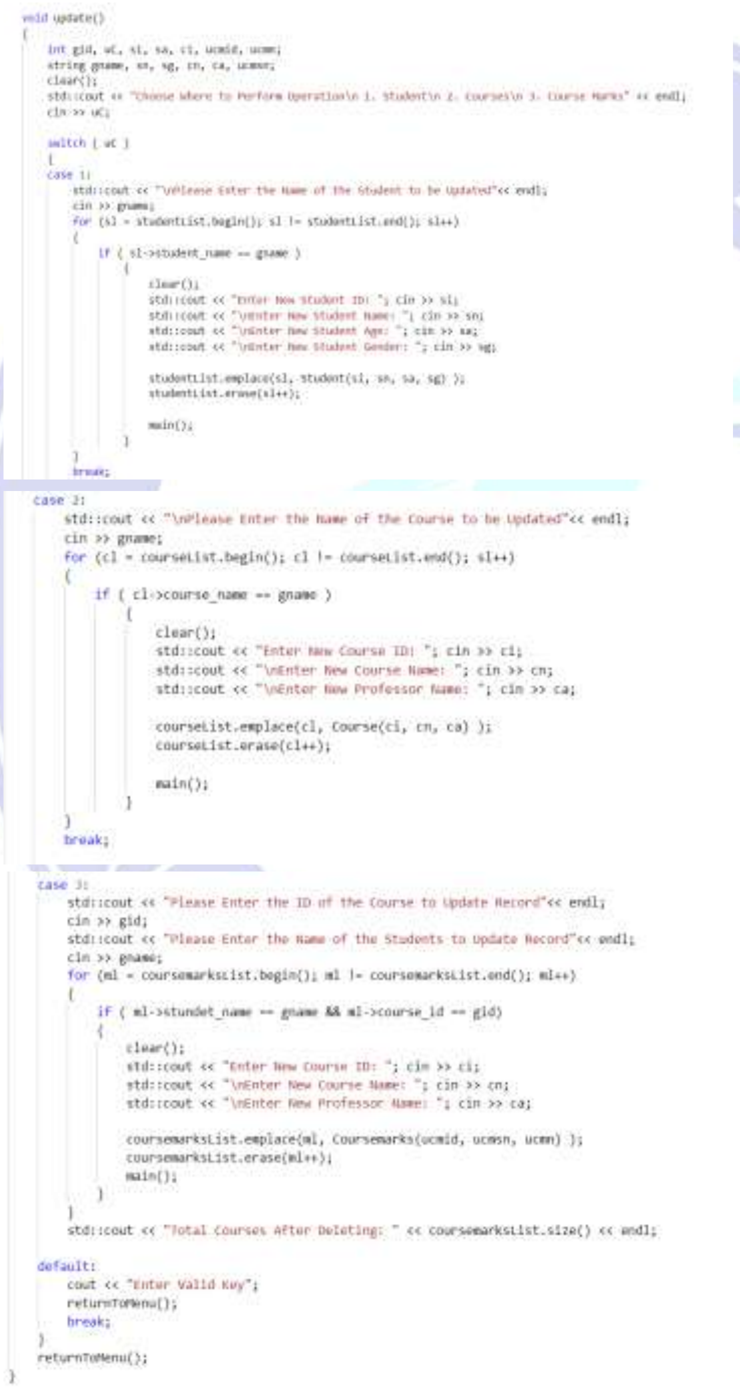
```cpp
case 2:
    std::cout << "Total Courses: " << courseList.size() << endl;
    std::cout << "Please Enter the Name of the Course to Delete"<< endl;
    cin >> gname;
    for (cl = courseList.begin(); cl != courseList.end(); cl++)
    {
        if ( cl->course_name == gname )
        {
            courseList.erase(cl);
            main();
        }
    }
    std::cout << "Total Courses After Deleting: " << courseList.size() << endl;
    break;
```



```cpp
case 3:
    std::cout << "Total Courses: " << coursemarksList.size() << endl;
    std::cout << "Please Enter the ID of the Course to Delete"<< endl;
    cin >> gid;
    std::cout << "Please Enter the Name of the Students to Delete"<< endl;
    cin >> gname;
    for (ml = coursemarksList.begin(); ml != coursemarksList.end(); ml++)
    {
        if ( ml->stundet_name == gname && ml->course_id == gid)
        {
            coursemarksList.erase(ml);
            main();
        }
    }
    std::cout << "Total Courses After Deleting: " << coursemarksList.size() << endl;
    break;

default:
    cout << "Enter Valid Key";
    returnToMenu();
    break;
}
returnToMenu();
}
```

Update info:

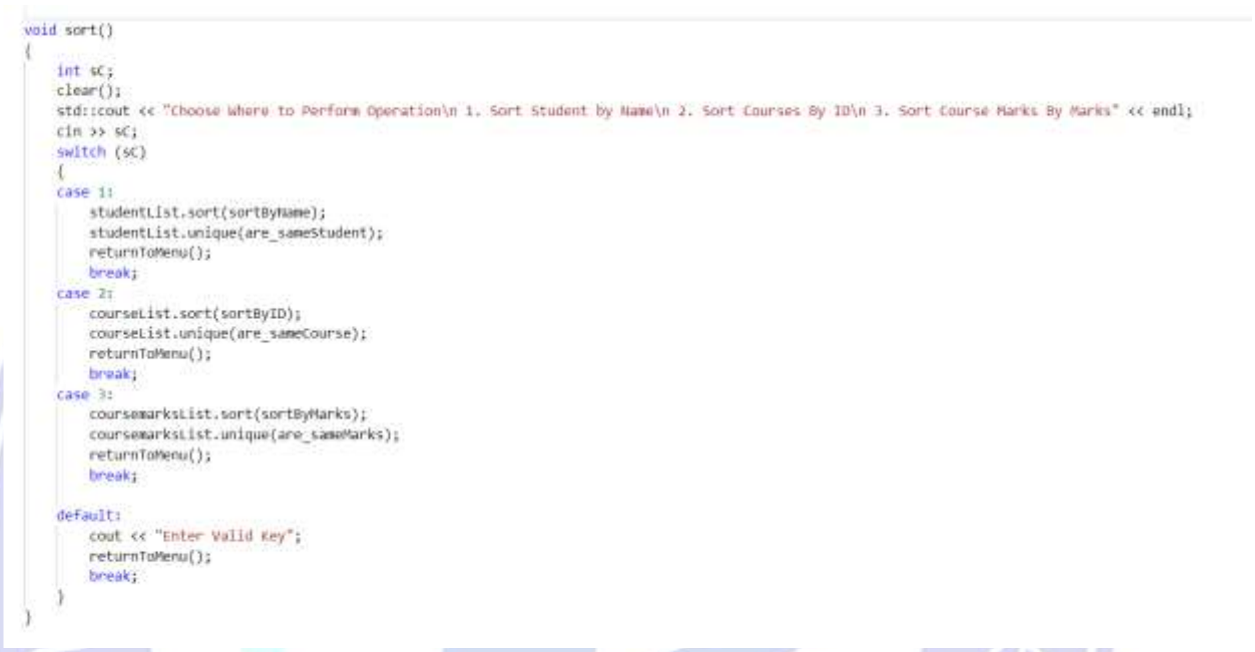
- Student
- Course
- Course Marks



```
void update()
{
    int gid, uC, sl, sa, ci, ucmid, ucmm;
    string gname, sn, sg, cn, ca, ucmsn;
    clear();
    std::cout << "Choose Where to Perform Operation\n 1. Student\n 2. Courses\n 3. Course Marks" << endl;
    cin >> uC;

    switch ( uC )
    {
    case 1:
        std::cout << "\nPlease Enter the Name of the Student to be Updated"<< endl;
        cin >> gname;
        for (sl = studentList.begin(); sl != studentList.end(); sl++)
        {
            if ( sl->student_name == gname )
            {
                clear();
                std::cout << "Enter New Student ID: "; cin >> sl;
                std::cout << "\nEnter New Student Name: "; cin >> sn;
                std::cout << "\nEnter New Student Age: "; cin >> sa;
                std::cout << "\nEnter New Student Gender: "; cin >> sg;

                studentList.emplace(sl, Student(sl, sn, sa, sg) );
                studentList.erase(sl++);

                main();
            }
        }
        break;
    case 2:
        std::cout << "\nPlease Enter the Name of the Course to be Updated"<< endl;
        cin >> gname;
        for (cl = courseList.begin(); cl != courseList.end(); sl++)
        {
            if ( cl->course_name == gname )
            {
                clear();
                std::cout << "Enter New Course ID: "; cin >> ci;
                std::cout << "\nEnter New Course Name: "; cin >> cn;
                std::cout << "\nEnter New Professor Name: "; cin >> ca;

                courseList.emplace(cl, Course(ci, cn, ca) );
                courseList.erase(cl++);

                main();
            }
        }
        break;
    case 3:
        std::cout << "Please Enter the ID of the Course to Update Record"<< endl;
        cin >> gid;
        std::cout << "Please Enter the Name of the Students to Update Record"<< endl;
        cin >> gname;
        for (ml = coursemarksList.begin(); ml != coursemarksList.end(); ml++)
        {
            if ( ml->stundet_name == gname && ml->course_id == gid)
            {
                clear();
                std::cout << "Enter New Course ID: "; cin >> ci;
                std::cout << "\nEnter New Course Name: "; cin >> cn;
                std::cout << "\nEnter New Professor Name: "; cin >> ca;

                coursemarksList.emplace(ml, Coursemarks(ucmid, ucmsn, ucmm) );
                coursemarksList.erase(ml++);
                main();
            }
        }
        std::cout << "Total Courses After Deleting: " << coursemarksList.size() << endl;

    default:
        cout << "Enter Valid Key";
        returnToMenu();
        break;
    }
    returnToMenu();
}
```

Sort Lists:

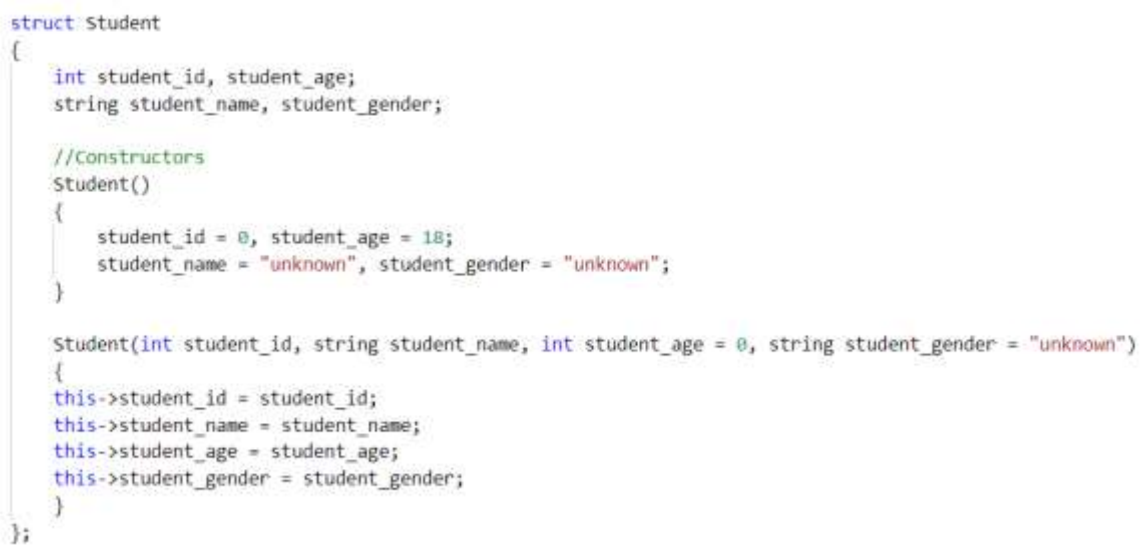
- Sort student by Name
- Sort Course by ID
- Sort Course Marks by Marks



```
void sort()
{
    int sC;
    clear();
    std::cout << "Choose Where to Perform Operation\n 1. Sort Student by Name\n 2. Sort Courses By ID\n 3. Sort Course Marks By Marks" << endl;
    cin >> sC;
    switch (sC)
    {
    case 1:
        studentList.sort(sortByName);
        studentList.unique(are_sameStudent);
        returnToMenu();
        break;
    case 2:
        courseList.sort(sortByID);
        courseList.unique(are_sameCourse);
        returnToMenu();
        break;
    case 3:
        coursemarksList.sort(sortByMarks);
        coursemarksList.unique(are_sameMarks);
        returnToMenu();
        break;

    default:
        cout << "Enter Valid Key";
        returnToMenu();
        break;
    }
}
```

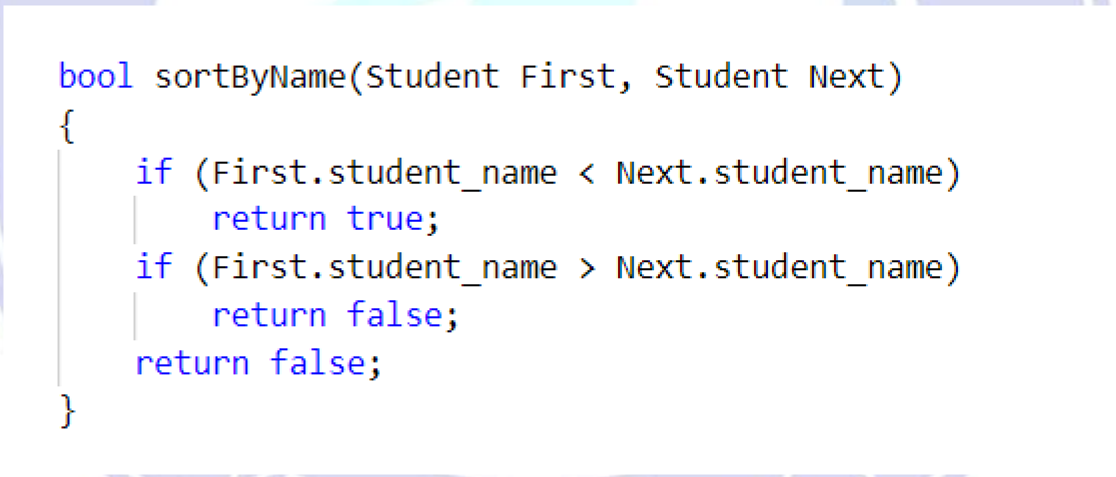
Exit:

Student class:



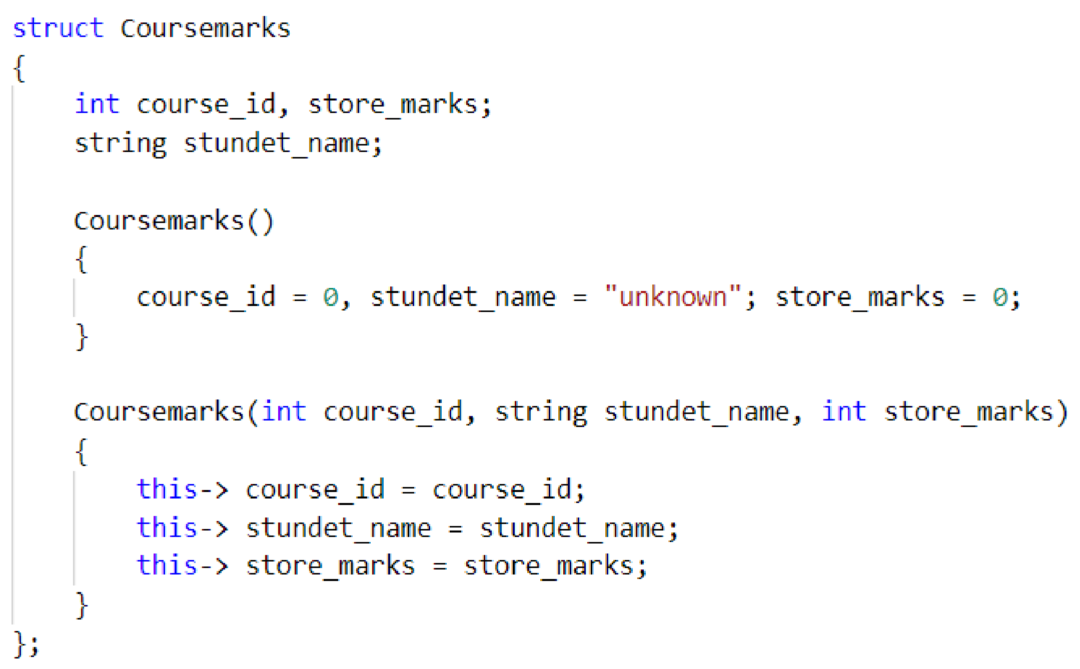
```
struct Student
{
    int student_id, student_age;
    string student_name, student_gender;

    //Constructors
    Student()
    {
        student_id = 0, student_age = 18;
        student_name = "unknown", student_gender = "unknown";
    }

    Student(int student_id, string student_name, int student_age = 0, string student_gender = "unknown")
    {
    this->student_id = student_id;
    this->student_name = student_name;
    this->student_age = student_age;
    this->student_gender = student_gender;
    }
};
```

Sort algorithm: **<u>Binary search Tree (log(n))</u>**



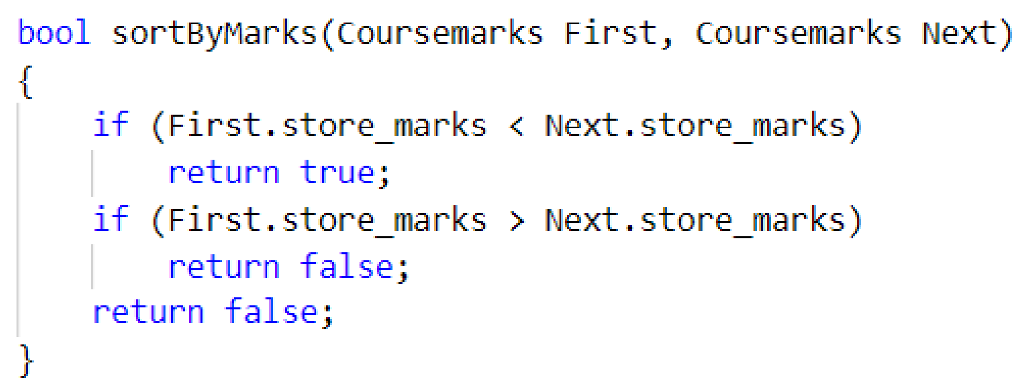
```
bool sortByName(Student First, Student Next)
{
    if (First.student_name < Next.student_name)
        return true;
    if (First.student_name > Next.student_name)
        return false;
    return false;
}
```

Course Marks Class:



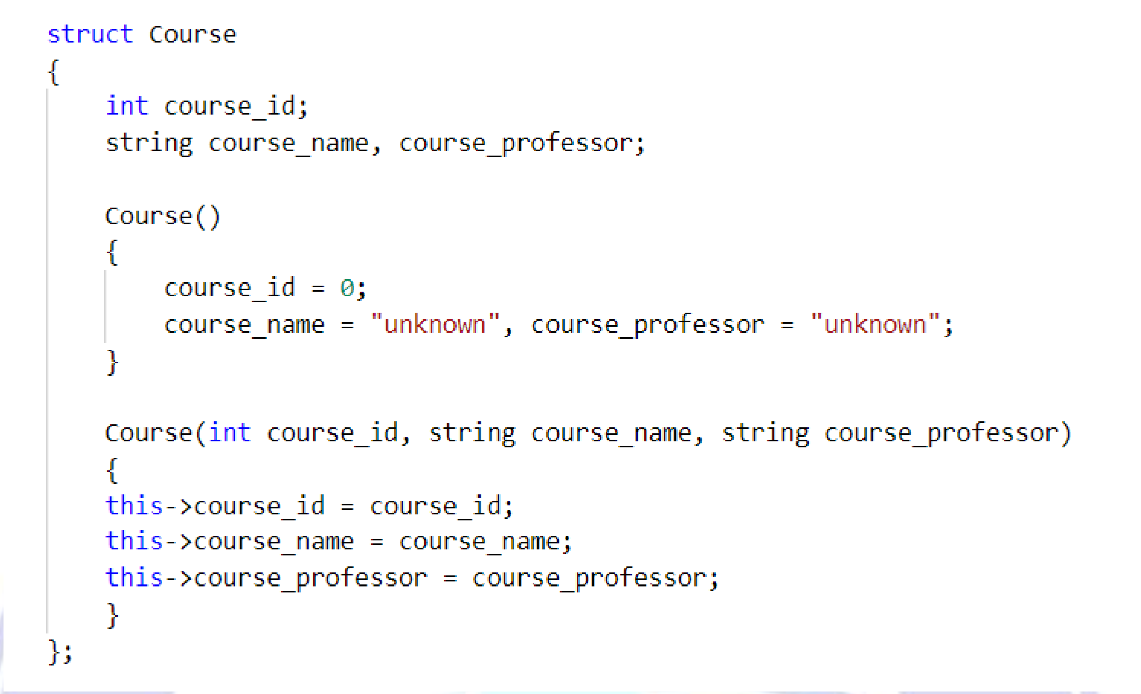
```
struct Coursemarks
{
    int course_id, store_marks;
    string stundet_name;

    Coursemarks()
    {
        course_id = 0, stundet_name = "unknown"; store_marks = 0;
    }

    Coursemarks(int course_id, string stundet_name, int store_marks)
    {
        this-> course_id = course_id;
        this-> stundet_name = stundet_name;
        this-> store_marks = store_marks;
    }
};
```

Sort algorithm: **<u>Binary search Tree (log(n))</u>**



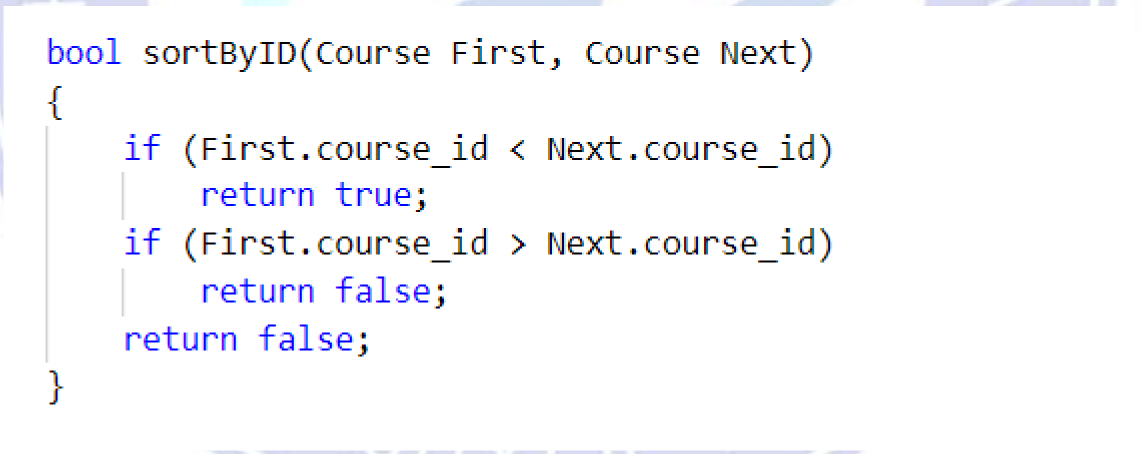
```
bool sortByMarks(Coursemarks First, Coursemarks Next)
{
    if (First.store_marks < Next.store_marks)
        return true;
    if (First.store_marks > Next.store_marks)
        return false;
    return false;
}
```

Course Class:



```
struct Course
{
    int course_id;
    string course_name, course_professor;

    Course()
    {
        course_id = 0;
        course_name = "unknown", course_professor = "unknown";
    }

    Course(int course_id, string course_name, string course_professor)
    {
    this->course_id = course_id;
    this->course_name = course_name;
    this->course_professor = course_professor;
    }
};
```

Sort algorithm: **Binary search Tree (log(n))**



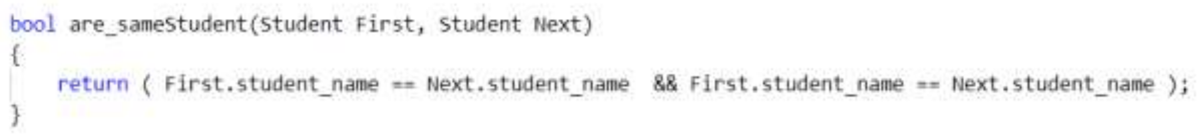
```
bool sortByID(Course First, Course Next)
{
    if (First.course_id < Next.course_id)
        return true;
    if (First.course_id > Next.course_id)
        return false;
    return false;
}
```

List Iterator:

```
std::list<Student> studentList;
std::list<Student>::iterator sl;

std::list<Course> courseList;
std::list<Course>::iterator cl;

std::list<Coursemarks> coursemarksList;
std::list<Coursemarks>::iterator ml;
```

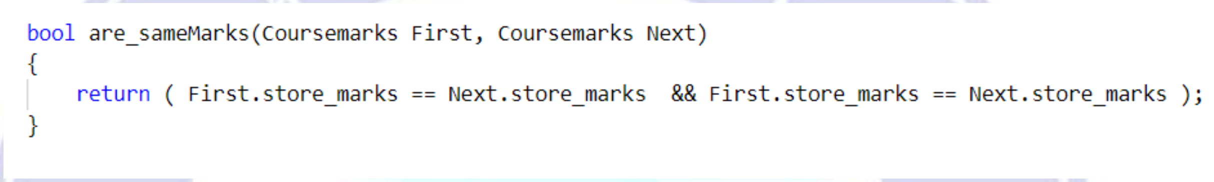
Bool Check for Same Instances

If Student has same name



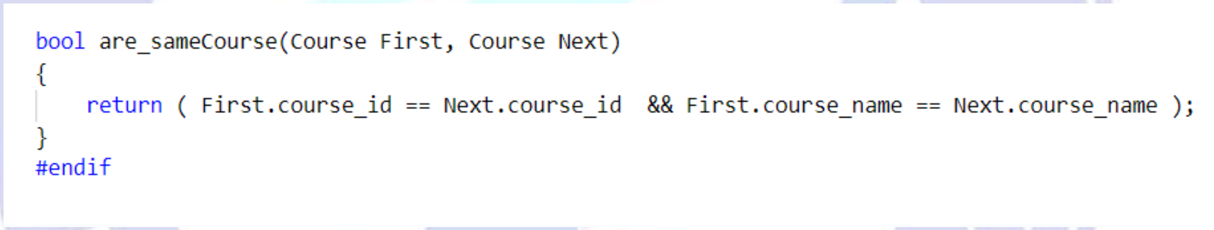
```
bool are_sameStudent(Student First, Student Next)
{
    return ( First.student_name == Next.student_name  && First.student_name == Next.student_name );
}
```

If Course-Marks has same marks



```
bool are_sameMarks(Coursemarks First, Coursemarks Next)
{
    return ( First.store_marks == Next.store_marks  && First.store_marks == Next.store_marks );
}
```

If course has same ID



```
bool are_sameCourse(Course First, Course Next)
{
    return ( First.course_id == Next.course_id  && First.course_name == Next.course_name );
}
#endif
```

Testing and analysis

The first step as to set up the launch.json file. As the program as built using vscode for windows using “bash” as default terminal. The configuration for debugging is provided below:-

```
{
   // Use IntelliSense to learn about possible attributes.
   // Hover to view descriptions of existing attributes.
   // For more information, visit:
https://go.microsoft.com/fwlink/?linkid=830387
   "version": "0.2.0",
   "configurations": [
      {
         "name": "gdb - Launch Debugger",
         "type": "cppdbg",
         "request": "launch",
         "program":
"${workspaceFolder}/src/${fileBasenameNoExtension}.exe",
         "args": [],
         "stopAtEntry": false,
         "cwd": "${workspaceFolder}",
         "environment": [],
         "externalConsole": true,
            "MIMode": "gdb",
            "miDebuggerPath": "C:/MinGW/bin/gdb.exe",
            "setupCommands": [
                {
                    "description": "Enable pretty-printing for gdb",
                    "text": "-enable-pretty-printing",
                    "ignoreFailures": true
                }
            ]
        },
    ]
}
```

Conclusion

It was an awesome learning experience for us while working on this project. This project took us through the various phases of project development. This report has discussed the development of a Student Management System. The objectives of this project were to develop the necessary software to show how the system works. Both objectives and all the requirements were met in detail. Also showed how software is designed. The system was realized by structured analysis and development methods. System runs well.

Bibliography

[1] cplusplus, ″cplusplus,″ cplusplus.com, 2020-2022. [Online]. Available: https://www.cplusplus.com/reference/list/list/. [Accessed 3 1 2021].

[2] ″TermColor | Github,″ Github, [Online]. Available: https://github.com/ikalnytskyi/termcolor. [Accessed 3 1 2021].

[3] ″GFeekForGeeks,″ GFeekForGeeks, [Online]. Available: https://www.geeksforgeeks.org/the-c-standard-template-library-stl/?ref=lbp. [Accessed 20 12 2020].

[4] X. Wu, ″Data Structures Course Slides,″ Hubei University OF Technology, Wuhan, Hubei China, 2020.