BỘ GIÁO DỤC VÀ ĐÀO TẠO

TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN – ĐHQG TPHCM

REPORT

Course: Programming Techniques – CSC10003

Project:

COURSE MANAGEMENT SYSTEM

Lecturers:

Đinh Bá Tiến

Trương Phước Lộc

Đỗ Nguyên Kha

REPORT PROJECT

TEAM 3

No	ID	Last name	First name
1	22127026	Ôn Gia	Bảo
2	22127123	Lê Hồ Phi	Hoàng
3	22127275	Trần Anh	Minh
4	22127402	Bế Lã Anh	Thư

Contents

1. The goal of the group:

The main goal our team is to provide a comprehensive and efficient online platform for universities to manage their course offerings. The system aims to streamline the administrative process and enhance communication between lecturers and students. It will allow lecturers to upload course materials, assignments, and grades while providing students with easy access to course information and resources. The system also aims to improve student-teacher interaction while enabling universities to monitor academic progress and performance. Ultimately, the goal is to create a platform that enhances the overall learning experience and contributes to academic success.

2. Implementation plan

Main programming language: C/C++
Main working environment: GitHub
Software for discussion: Messenger, Google Meet
Workflow: Each team member is allocated particular duties that contribute to the completion of the project. These tasks need to be fulfilled and uploaded to GitHub on a weekly basis. It is within the team's guidelines for members to assist each other in completing their tasks. However, members need to be accountable for their own contributions. In case of coding errors or mishaps, the individual who made the final edit will need to take responsibility.
Task assignment table:

No	ID	Last name	First name	Task assignment
1	22127026	Ôn Gia	Bảo	Task 14, 15, 16, 17, 18, 24, files .csv, write report
2	22127123	Lê Hồ Phi	Hoàng	Task 19, 20, 21, 22, 23, merge code, write report

3	22127275	Trần Anh	Minh	Task 6, 7, 8, 9, 10, 11, 12, 13, merge code, UI, write code
4	22127402	Bế Lã Anh	Thư	Task 1, 2, 3, 4, 5

3. Description of Structures Used

A. Libraries:

```
#include <iostream>
#include <string>
#include <cstring>
#include <fstream>
#include <iomanip>
#include <vector>
#include <conio.h>
#include <Windows.h>
#include <sstream>
#include <cctype>
#include <algorithm>
```

B. Struct:

```
struct Account
{
    string username = "", password = "";
    string firstName = "", lastName = "";
    string Gender = ""; // M is male, F is female
    string SocialID = "";
    string birth = "";
    int role; // 1 student, 2 teacher, 3 staff.
    Account *next = nullptr, *prev = nullptr;
};
```

The Account struct is designed to represent user accounts in the management system. It contains the following member variables:

- username: a string representing the username of the account.
- password: a string representing the password of the account.

- firstName: a string representing the first name of the account holder.
- lastName: a string representing the last name of the account holder.
- Gender: a string representing the gender of the account holder (M for male, F for female).
- SocialID: a string representing the social ID of the account holder.
- birth: a string representing the date of birth of the account holder (DD/MM/YYYY).
- role: an integer representing the role of the account holder (1 is student, 2 is teacher, 3 is staff).
- next: a pointer to the next Account struct in a linked list of accounts.
- prev: a pointer to the previous Account struct in a linked list of accounts.

The Account struct provides a centralized and structured way to manage and authenticate user information, ensuring secure access and appropriate permissions to different resources within the course-management system.

```
struct ScoreBoardCourse
{
    double totalMark = 0, finalMark = 0, midMark = 0, otherMark = 0;
};
```

The ScoreBoardCourse struct is used in a course-management system to store and manage the marks of a student in different assessments, including the final exam, mid-term exam, and other assignments. The struct has four double-type variables, including totalMark, finalMark, midMark, and otherMark. The totalMark variable stores the overall mark of the student in the course, whereas the finalMark, midMark, and otherMark variables store the marks obtained by the student in the final exam, mid-term exam, and other assessments, respectively.

```
// student của một course
struct StudentCourse
{
    string ID = "";
    string FullName = "";
    StudentCourse *next = nullptr, *prev = nullptr;
    ScoreBoardCourse ScoreBoardCourse;
};

```

The StudentCourse struct is designed to represent an student in a course.

- ID: a string represent the ID of the student
- FullName: a string represent the student's full name
- next: a pointer to the next StudentCourse in a linked list of students in a course
- prev: a pointer to the prev StudentCourse in a linked list of student in a course
- ScoreBoardCourse: store score board of a student in a course

By using a linked list to connect each StudentCourse object, it allows for efficient traversal and manipulation of the data structure. The inclusion of the ScoreBoardCourse object also allows for the storage and calculation of relevant course metrics, such as average scores and grading information. Overall, this struct plays a crucial role in organizing and managing student and course data within an education system.

```
struct Course
{
    string Name = "";
    string CourseID = ""; // eg CSC10002
    int Credits = 0, maxStudents = 70, numStudents = 0;
    string Room = "";
    string TeacherName = "";
    string Day = "";      // eg MON TUE...
    string Session = ""; // eg S1 then session = 1
    StudentCourse *studentCourse = nullptr;
    Course *next = nullptr, *prev = nullptr;
};
```

The Course struct is designed to represent a course in a semester.

- Name: a string that represents the name of the course.
- courseID: a string that uniquely identifies a course in the university's course catalog (eg CSC10002).
- Credits - an integer value that represents the number of credits for the course.
- maxStudents - an integer value that represents the maximum number of students that can be enrolled in the course.
- numStudents - an integer value that represents the number of students who are currently enrolled in the course.
- Room: a string that represents where the course take place (eg I62).
- teacherName - a string that represents the name of the teacher who is teaching the course.
- Day: a string that represents the day of the week on which the course is scheduled (eg MON, TUE, …).
- Session: a string that represents the time period during which the course is scheduled (eg S1 then session = 1)
- studentCourse: a pointer to the first student in the linked list of enrolled students.
- next: a pointer to the next course in the linked list of courses.
- prev: a pointer to the prev course in the linked list of courses.

The "Course" structure is a key element of software that manages student records in an academic institution, holding specific details about a course including the course name, ID, teacher name, session, credits, day of the week, and a roster of enrolled students.

```
struct CourseStudent
{
    Course *course = nullptr;
    CourseStudent *next = nullptr;
};
```

This CourseStudent struct represents a node in a linked list of courses that a student is enrolled in.

- course: a pointer to a course of that student.
- next: a pointer to the next CourseStudent node in a linked list, which is the next course of that student.

This linked list structure is useful for storing and maintaining a list of course that a student is enrolled in, allowing for easy iteration and modification of this list as needed.
Overall, the CourseStudent struct is a simple but effective way to represent the relationship between courses and enrolled students in the course management system.

```
struct Student
{
    Account *accStudent = nullptr;
    string ID = "";
    string ClassName = "";
    CourseStudent *course = nullptr;
    Student *next = nullptr, *prev = nullptr;
};
```

The 'Student' struct serves as a representation of a real-life student in the context of a university and stored in a specific class. It is comprised of the following components:

- accStudent: This pointer stores the address of the student's account. The account stores the student's username, password, and other essential details for authentication and personalization in the system.
- ID: a string represents the student's identification number, which helps distinguish each student from one another and acts as a unique identifier in the system. It allows the software to quickly access, search, and update information pertaining to individual students.
- ClassName: a string that represents the class name that the student belongs to.
- course: a pointer to the CourseStudent structure, which comprises the courses the student has enrolled in during the semester. It enables management of a student's curriculum, including the updating and accessing of course details.
- Student *next, *prev: two pointers to the adjacent Student structs in a double-linked list. 'next' is a pointer to the subsequent student, while 'prev' is a pointer to the preceding student on the list of students in a

specific class. This facilitates navigating the student list and performing various actions related to student data storage and organization.

The 'Student' struct is a well-designed structure that efficiently represents students within the course management system software. It facilitates streamlined data storage, organization, and management and ensures scalability and flexibility within the system. This struct plays a significant role in the overall performance of the software, contributing to its success in managing the educational work of teachers and enabling students to control their studies effectively.

```
struct Class
{
    string Name; // eg 22CLC02
    Student *StudentClass = nullptr;
    Class *next = nullptr, *prev = nullptr;
};
```

In creating a software application for the management of university students, the Class struct plays a crucial role.

- Name: a string representing the unique name for each class(e.g., "22CLC02").
- StudentClass: a pointer to a Student struct, which corresponds to the head of a linked list representing the collection of students enrolled in the Class. This enables efficient student enrollment, search, and removal operations within the class.
- next and prev: These attributes are pointers to the next and previous Class structs in a doubly linked list, enabling easy navigation between classes and efficient implementation of class-based operations like insertion, removal, and search.

As part of the bigger functional ecosystem of the software, the Class struct contributes significantly to streamlining academic workflows, improving resource utilization, and enhancing overall user experience for students, teachers, and administrators.

```
struct Semester
{
    int No; // eg semester 1 thì No = 1
    int Year; // only the start year, when cout
    string startDate, endDate;
    Course *course = nullptr;
    Semester *next = nullptr, *prev = nullptr;
};
```

The Semester struct is a necessary data structure for managing academic semesters in the management system.

- No: a integer number represents the semester number.
- Year: a integer number represents the school year it belongs to (only the start year).
- startDate, endDate: strings represent the dates when the semester begins and finishes.
- course: a pointer to a linked list of courses in that semester.
- next: a pointer to the next semester in the same school year.
- prev: a pointer to the previous semester in the same school year.

The Semester struct is used to represent a single semester in a course management system. Overall, the Semester struct provides a convenient way to organize and manage information about individual semesters in a course management system. The use of a linked list for storing multiple semesters allows for easy traversal and manipulation of the data.

```
struct Year
{
    int yearStart;
    Class *Class = nullptr;
    Semester *NoSemester = nullptr;
    Year *next = nullptr;
};
```

The struct Year represents a year in the course-management-system project. It has the following data members:

- yearStart: an integer that indicates the starting year of this year in the academic calendar.
- Class: a pointer to a Class object that represents the classes existed in this year. (such as 22CLC02, …).
- NoSemester: a pointer to the linked list of semesters in that year. There are 3 semesters in each year.
- next: a pointer to the next year object in the linked list of years. This is used to iterate over all the years in the academic calendar. It is initially set to nullptr.
- prev: a pointer to the previous year object in the linked list of years. This is used to navigate backward in the linked list. It is initially set to nullptr.

Overall, the Year struct is a convenient way to organize years in the academic calendar of the course-management-system project. It allows for efficient iteration and navigation through a linked list of years, and provides pointers to the courses taught in each year.

C. Main components

i. [All] Section

- File: [All]_Check_Validity.cpp: This file contains functions to check whether the value is valid or not
  - The "CapitalClassName" function changes the class name string into the correct form (22clc02 -> 22CLC02).

```
//? Turn the classname to uppercase
void CapitalClassName(string &name)
{
    for (int i = 0; i < name.length(); i++)
        name[i] = toupper(name[i]);
}
```

  - "isValidStudentID" function checks if the student ID has 8 characters and the first 2 characters fit the year.

```
//? Return true if the studentID is valid
bool isValidStudentID(string studentID, string year)
{
    if (studentID.length() != 8)
        return false;
    for (int i = 0; i < 8; i++)
        if (!isdigit(studentID[i]))
            return false;
    if (studentID.substr(0, 2) != year)
        return false;
    return true;
}
```

- "checkYear" check whether the newly created year existed or not.

```
//? Return true if the year existed
bool checkYear(Year *curYear, int year)
{
    while (curYear)
    {
        if (curYear->yearStart == year)
            return true;
        curYear = curYear->next;
    }
    return false;
}
```

- "checkClass" check whether a newly created class existed or not.

```
//? Return true if the class already exists
bool checkClass(Year *curYear, string className)
{
    Class *curClass = curYear->Class;
    while (curClass)
    {
        if (curClass->Name == className)
            return true;
        curClass = curClass->next;
    }
    return false;
}
```

- "Check_Student" check whether a newly created student existed or not.

```
//? Return true if the student already exists
bool Check_Student(Class *curClass, string studentID)
{
    Student *curStudent = curClass->StudentClass;
    while (curStudent)
    {
        if (curStudent->ID == studentID)
            return true;
        curStudent = curStudent->next;
    }
    return false;
}
```

- "isLeap" is to check if the year is a leap.

```
//? return true if it is a leap year
bool isLeap(int year)
{
    return (year % 4 == 0 && (year % 100 != 0 || year % 400 == 0));
}
```

- "isValidDate" to check the validity of an inputting date.

```
//? return true if the date is valid
bool isValidDate(const string &dateOfBirth)
{
    if (dateOfBirth.length() != 10)
        return false;

    if (dateOfBirth[2] != '/' || dateOfBirth[5] != '/')
        return false;

    int dd = stoi(dateOfBirth.substr(0, 2));
    int mm = stoi(dateOfBirth.substr(3, 2));
    int yyyy = stoi(dateOfBirth.substr(6, 4));

    if (mm < 1 || mm > 12)
        return false;

    int daysInMonth[] = {31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};
    if (isLeap(yyyy))
        daysInMonth[1] = 29;

    if (dd < 1 || dd > daysInMonth[mm - 1])
        return false;

    return true;
}
```

- "isDay" checks the string if it is in MON, TUE, WED, THU, FRI, SAT, SUN.

```
//? return true if the Day format is correct
bool isDay(string day)
{
    return (day == "MON" || day == "TUE" || day == "WED" || day == "THU" || day == "FRI" || day == "SAT" || day == "SUN");
}
```

- "isSession" to check if it is in S1, S2, S3, S4.

```
//? return true if the session format is correct
bool isSession(string ss)
{
    return (ss == "S1" || ss == "S2" || ss == "S3" || ss == "S4");
}
```

- "isValidGender" check input is M or F.

```
//? true if it is in the correct format
bool isValidGender(const string &gender)
{
    return (gender == "F" || gender == "M");
}
```

- "isValidYear" checks the format of the school year is yyyy_yyyy.

```
//? true if it is correct format (yyyy_yyyy)
bool isValidYear(string year)
{
    if (year.size() != 9)
        return false;
    else if (year[4] != '-' && year[4] != '_')
        return false;
    int y1 = stoi(year.substr(0, 4));
    int y2 = stoi(year.substr(5, 4));
    if (y1 < 0 || y2 < 0)
        return false;
    return (y2 - y1 == 1);
}
```

- "SeparateName" separates the full name into first and last name.

```
void SeparateName(string full, string &first, string &last)
{
    size_t pos = full.find_last_of(" ");
    first = full.substr(0, pos);
    last = full.substr(pos + 1);
}
```

- File: [All]_Del_LL.cpp: This file consists of functions that delete all the created linked list
  - "DeleteYear" function deletes every node of the linked list and the class with the student node inside it.

```
//? Delete the year linked list
//! Lồng delete student course vô trong lun
void DeleteYear(Year *&yearHead)
{
    Year *yearTMP = yearHead;
    while (yearTMP)
    {
        // Class in the year
        Class *ClassTMP = yearTMP->Class;
        while (ClassTMP)
        {
            // Delete all the slots for student in linked list
            Student *StudentTMP = ClassTMP->StudentClass;
            while (StudentTMP)
            {
                Student *StudentTMP2 = StudentTMP;
                StudentTMP = StudentTMP->next;
                delete StudentTMP2;
            }
            Class *ClassTMP2 = ClassTMP;
            ClassTMP = ClassTMP->next;
            delete ClassTMP2;
        }
        // Delete all semesters in the year
        DeleteSMT(yearTMP);
        Year *yearTMP2 = yearTMP;
        yearTMP = yearTMP->next;
        delete yearTMP2;
    }
    yearHead = nullptr; // Set the head to nullptr
}
```

- "DeleteStudent" erase all instance of students in the linked list inside courses in the semester.

```
void DeleteStudent(StudentCourse *&stud_head)
{
    if (!stud_head)
        return;
    DeleteStudent(stud_head->next); // Recursion to delete the next student slot
    delete stud_head;
    stud_head = nullptr;
}
```

- "DeleteCourse" erase all instance of courses in the linked list inside the semester.

```
void DeleteCourse(Course *&course_head)
{
    if (!course_head)
        return;
    DeleteCourse(course_head->next); // Recursion to delete the next course inside
    DeleteStudent(course_head->studentCourse); // Delete the linked list that store student in4
    delete course_head;
    course_head = nullptr; // return the head to nullptr
}
```

- "DeleteSMT" delete all semester in a current year and is put inside the "DeleteYear" function.

```
void DeleteSMT(Year *yearHead)
{
    if (!yearHead->NoSemester)
        return;
    Semester *semHead = yearHead->NoSemester;
    while (semHead)
    {
        DeleteCourse(semHead->course); // Call the delete the course inside semester
        Semester *tmp = semHead->next;
        delete semHead;
        semHead = tmp;
    }
    yearHead->NoSemester = nullptr; // Return the semester to nullptr
}
```

- "DelAccount" cleans up the space that the account linked list used.

```
//? Delete the account linked list
void DelAccount(Account *&accHead)
{
    if (!accHead)
        return;
    while (accHead)
    {
        Account *tmp = accHead->next;
        delete accHead;
        accHead = tmp;
    }
    accHead = nullptr;
}
```

- File: [All]_Draw.cpp: contains all functions with the usage of drawing out the information that users can use the arrow button to move around.
  - The "Draw_XY" takes in the list of options in order to print out the list of actions for the user to choose.

```cpp
int Draw_XY(vector<string> menu, int xStart, int yStart, int nOption_eachTime, int width, int color)
{
    if (nOption_eachTime > menu.size())
        nOption_eachTime = menu.size();
    int amount_page = ceil(menu.size() * 1.0 / nOption_eachTime), num_page = 1, tmp;
    char arrow;
    vector<int> choice(menu.size(), 0);
    int cur = 0;
    int step = 0;
    bool stop = false;
    while (!stop)
    {
        if (step < (cur / nOption_eachTime) * nOption_eachTime)
        {
            for (int i = 0; i < nOption_eachTime; i++)
            {
                TextColor(7);
                for (int j = 0; j < 3; j++)
                {
                    goToXY(xStart, yStart + i * 3 + j);
                    cout << setw(width) << " ";
                }
            }
        }
        step = (cur / nOption_eachTime) * nOption_eachTime;
        choice[cur] = 1;
        for (int i = 0; i < nOption_eachTime && i + step < menu.size(); i++)
        {
            int tmp = (choice[i + step] ? color : 7);

            TextColor(tmp);
            for (int j = 0; j < 3; j++)
```

```cpp
        for (int i = 0; i < nOption_eachTime && i + step < menu.size(); i++)
        {
            int tmp = (choice[i + step] ? color : 7);

            TextColor(tmp);
            for (int j = 0; j < 3; j++)
            {
                goToXY(xStart, yStart + i * 3 + j);
                cout << setw(width) << " ";
            }
            goToXY(xStart + 2, yStart + 1 + i * 3);
            cout << menu[i + step];
        }
        TextColor(07);
        goToXY(xStart, yStart + nOption_eachTime * 3 + 1);
        cout << setw(20) << " ";
        tmp = min((int)(menu.size() - step), nOption_eachTime);

        num_page = ceil((cur + 1) * 1.0 / nOption_eachTime);
        goToXY(xStart, yStart + tmp * 3 + 1);
        if (amount_page > 1)
        {
            cout << "  " << (num_page > 1 ? char(30) : char(32))
                 << " " << num_page << "/" << amount_page
                 << " " << (num_page < amount_page ? char(31) : char(32));
        }
        if (tmp = _getch())
        {
            switch (tmp)
            {
            case ENTER:
                stop = true;
```

```cpp
        if (tmp = _getch())
        {
            switch (tmp)
            {
            case ENTER:
                stop = true;
                break;
            case UP:
                choice[cur] = 0;
                cur = (cur > 0 ? cur - 1 : menu.size() - 1);
                break;
            case DOWN:
                choice[cur] = 0;
                cur = (cur < menu.size() - 1 ? cur + 1 : 0);
                break;
            }
        }
    }
    for (int i = 0; i < nOption_eachTime; i++)
    {
        TextColor(7);
        for (int j = 0; j < 3; j++)
        {
            goToXY(xStart, yStart + i * 3 + j);
            cout << setw(width) << " ";
        }
    }
    goToXY(xStart, yStart + nOption_eachTime * 3 + 1);
    cout << setw(20) << " ";
    tmp = min((int)(menu.size() - step), nOption_eachTime);
    return cur;
}
```

- The "Draw_table" takes in the 2d array to draw onto the console a table with its data passed in.

```
void Draw_table(string **table, string *title, int num_row, int num_col, int *width, int height,
                int x, int y, int Row_eachTime, int Col_eachTime,
                bool *edit_Col, int &x_cur, int &y_cur)
{
    if (Col_eachTime > num_col)
        Col_eachTime = num_col;
    if (Row_eachTime > num_row)
        Row_eachTime = num_row;
    int **choice = new int *[num_row];
    for (int i = 0; i < num_row; i++)
    {
        choice[i] = new int[num_col];
        for (int j = 0; j < num_col; j++)
            choice[i][j] = 0;
    }
    int *pos = new int[num_col]; // vị trí bắt đầu của các cột, bắt đầu từ cột đầu là 0
    pos[0] = 0;
    for (int i = 1; i < num_col; i++)
    {
        if (i % Col_eachTime == 0)
            pos[i] = 0;
        else
            pos[i] = pos[i - 1] + width[i - 1];
    }

    int x_prev = num_col, y_prev = num_row,
        x_step, y_step,
        num_page_hor, num_page_ver,
        amount_page_hor = ceil(num_col * 1.0 / Col_eachTime), amount_page_ver = ceil(num_row * 1.0 / Row_eachTime);
    bool stop = false;

    x_step = (x_cur / Col_eachTime) * Col_eachTime;
```

```
    x_step = (x_cur / Col_eachTime) * Col_eachTime;
    y_step = (y_cur / Row_eachTime) * Row_eachTime;
    choice[y_cur][x_cur] = 1;
    while (stop != true)
    {
        if (y_prev / Row_eachTime != y_cur / Row_eachTime || x_prev / Col_eachTime != x_cur / Col_eachTime)
        {
            TextColor(7);
            // xóa kẻ đầu tiên
            for (int j = 0; j < Col_eachTime && j + x_step < num_col; j++)
            {
                goToXY(x + pos[j + x_step], y - 1);
                cout << setw(width[j + x_step] + 1) << " ";
            }
            // xóa ô tiêu đề và các ô trước
            for (int i = 0; i <= Row_eachTime && i + y_step <= num_row; i++)
            {
                for (int j = 0; j < Col_eachTime && j + x_step < num_col; j++)
                {

                    goToXY(x + pos[j + x_step], y + i * (height + 1));
                    cout << setw(width[j + x_step] + 1) << " ";
                    goToXY(x + pos[j + x_step], y + i * (height + 1) + 1);
                    cout << setw(width[j + x_step] + 1) << " ";
                }
            }
            goToXY(x, y + 2 + min(Row_eachTime, num_row - y_step) * (height + 1));
            cout << setw(20) << " ";
            x_step = (x_cur / Col_eachTime) * Col_eachTime;
            y_step = (y_cur / Row_eachTime) * Row_eachTime;
            goToXY(x, y - 1);
            cout << char(218); // 218 is top left
```

```
        cout << char(218); // 218 is top left
        goToXY(x, y);
        cout << char(179);
        goToXY(x, y + 1);
        cout << char(195);
        for (int j = 0; j < Col_eachTime && j + x_step < num_col; j++)
        {
            for (int t = 0; t < width[j + x_step] - 1; t++)
            {
                goToXY(x + pos[j + x_step] + t + 1, y - 1);
                cout << char(196);
                goToXY(x + pos[j + x_step] + t + 1, y + 1);
                cout << char(196);
            }
            goToXY(x + pos[j + x_step] + 2, y);
            cout << title[j + x_step];
            if (j == Col_eachTime - 1 || j == num_col - x_step - 1)
            {
                goToXY(x + pos[j + x_step] + width[j + x_step], y - 1);
                cout << char(191);
                goToXY(x + pos[j + x_step] + width[j + x_step], y + 1);
                cout << char(180);
            }
            else
            {
                goToXY(x + pos[j + x_step] + width[j + x_step], y + -1);
                cout << char(194);
                goToXY(x + pos[j + x_step] + width[j + x_step], y + 1);
                cout << char(197);
            }
            goToXY(x + pos[j + x_step] + width[j + x_step], y);
            cout << char(179);

            goToXY(x + pos[j + x_step] + width[j + x_step], y);
            cout << char(179);
        }

        y += 2;
        choice[y_cur][x_cur] = 1;
        for (int i = 0; i < Row_eachTime && i + y_step < num_row; i++)
        {
            for (int j = 0; j < Col_eachTime && j + x_step < num_col; j++)
            {
                goToXY(x + pos[j + x_step] + 1, y + i * (height + 1) + 1);
                for (int t = 0; t < width[j + x_step] - 1; t++)
                    cout << char(196);
                goToXY(x + pos[j + x_step], y + i * (height + 1) + 1);
                if (i != Row_eachTime - 1 && i != num_row - 1 - y_step)
                {
                    if (j == 0)
                        cout << char(195); // 195  is|-
                    else
                        cout << char(197); // 197 is +
                }
                else
                {
                    if (j == 0)
                    {
                        cout << char(192); // 192  is|_

                        // đếm số trang cho bảng
                        goToXY(x, y + i * (height + 1) + 2);
                        if (amount_page_hor > 1)
                        {
                            num_page_hor = ceil((x_cur + 1) * 1.0 / Col_eachTime);
                            if (amount_page_hor > 1)
                            {
                                num_page_hor = ceil((x_cur + 1) * 1.0 / Col_eachTime);
                                cout << "  " << (num_page_hor > 1 ? char(17) : char(32))
                                     << " " << num_page_hor << "/" << amount_page_hor
                                     << " " << (num_page_hor < amount_page_hor ? char(16) : char(32));
                            }
                            if (amount_page_ver > 1)
                            {
                                num_page_ver = ceil((y_cur + 1) * 1.0 / Row_eachTime);
                                cout << "  " << (num_page_ver > 1 ? char(30) : char(32))
                                     << " " << num_page_ver << "/" << amount_page_ver
                                     << " " << (num_page_ver < amount_page_ver ? char(31) : char(32));
                            }
                        }
                        else if (j != Col_eachTime - 1 && j != num_col - x_step - 1)
                            cout << char(193); // 193 is _|_
                        // in số trang ở cuối bảng
                    }

                    goToXY(x + pos[j + x_step], y + i * (height + 1));
                    cout << char(179); // 179 is |

                    int tmp = (choice[i + y_step][j + x_step] ? 0xF3 : 7);
                    TextColor(tmp);
                    goToXY(x + 1 + pos[j + x_step], y + i * (height + 1)); // +1 để in background không bị đè
                    cout << setw(width[j + x_step] - 1) << " ";
                    goToXY(x + 2 + pos[j + x_step], y + i * (height + 1)); //+3 là để lùi vào trong một ít cho đẹp
                    cout << table[i + y_step][j + x_step];

                    TextColor(7);
                    goToXY(x + pos[j + x_step] + width[j + x_step], y + i * (height + 1));
```

```
                TextColor(7);
                goToXY(x + pos[j + x_step] + width[j + x_step], y + i * (height + 1));
                cout << char(179); // 179 is |
                goToXY(x + pos[j + x_step] + width[j + x_step], y + i * (height + 1) + 1);
                if (i == Row_eachTime - 1 || i == num_row - 1 - y_step)
                    cout << char(217); // 217 is _|
                else
                    cout << char(180); // 180 -|
            }
        }
    }
    else
    {
        y += 2;
        TextColor(7);
        goToXY(x + 1 + pos[x_prev], y + (y_prev % Row_eachTime) * (height + 1));
        cout << setw(width[x_prev] - 1) << " ";
        goToXY(x + 2 + pos[x_prev], y + (y_prev % Row_eachTime) * (height + 1));
        cout << table[y_prev][x_prev];

        TextColor(0xF3);
        goToXY(x + 1 + pos[x_cur], y + (y_cur % Row_eachTime) * (height + 1));
        cout << setw(width[x_cur] - 1) << " ";
        goToXY(x + 2 + pos[x_cur], y + (y_cur % Row_eachTime) * (height + 1));
        cout << table[y_cur][x_cur];
    }

    y -= 2;
    x_prev = x_cur;
    y_prev = y_cur;
    TextColor(7);
    int tmp;
    if (tmp = _getch())
    {
        switch (tmp)
        {
        case 27:
            x_cur = -1;
            y_cur = -1;
            stop = true;
            break;
        case ENTER:
            if (edit_Col[x_cur] == false)
                Message_Warning("This cell is not editable", "Error!");
            else
                stop = true;
            break;
        case UP:
            choice[y_cur][x_cur] = 0;
            y_cur = (y_cur > 0 ? y_cur - 1 : num_row - 1);
            break;
        case DOWN:
            choice[y_cur][x_cur] = 0;
            y_cur = (y_cur < num_row - 1 ? y_cur + 1 : 0);
            break;
        case LEFT:
            choice[y_cur][x_cur] = 0;
            x_cur = (x_cur > 0 ? x_cur - 1 : num_col - 1);
            break;
        case RIGHT:
            choice[y_cur][x_cur] = 0;
            x_cur = (x_cur < num_col - 1 ? x_cur + 1 : 0);
        }
    }
}
for (int i = 0; i < num_row; i++)
    delete[] choice[i];
delete[] choice;
delete[] pos;
}
```

- "Draw_Horizontal_XY" is another version of Draw_XY but it is used to print out the table in the horizontal ways.

```
int Draw_Horizontal_XY(vector<string> menu, int x, int y, int &cur, int color)
{
    vector<int> choice(menu.size(), 0);
    while (1)
    {
        choice[cur] = 1;
        for (int i = 0; i < choice.size(); i++)
        {
            for (int j = 0; j < 3; j++)
            {
                int tmp = (choice[i] ? color : 7);
                TextColor(tmp);
                goToXY(x + 3 * i, y + j);
                cout << "   ";
                goToXY(x + 1 + 3 * i, y + 1);
                cout << menu[i];
            }
        }
        TextColor(07);
        int tmp;
        if (tmp = _getch())
        {
            switch (tmp)
            {
            case ENTER:
                system("cls");
                return cur;
            case LEFT:
                choice[cur] = 0;
                cur = (cur > 0 ? cur - 1 : menu.size() - 1);
                break;
            case RIGHT:
                choice[cur] = 0;
                cur = (cur < menu.size() - 1 ? cur + 1 : 0);
                break;
            }
        }
    }
}
```

- "Draw_ShortVer" is another version of "Draw_XY" but the printout list only covers 1 block instead of 3.

```
int Draw_ShortVer(vector<string> menu, int x, int y, int color)
{
    vector<int> choice(menu.size(), 0);
    int cur = 0;
    while (1)
    {
        choice[cur] = 1;
        for (int i = 0; i < choice.size(); i++)
        {
            int tmp = (choice[i] ? color : 7);

            TextColor(tmp);
            goToXY(x, y + i);
            cout << "                          ";
            goToXY(x + 2, y + i);
            cout << menu[i];
        }
        TextColor(07);
        int tmp;
        if (tmp = _getch())
        {
            switch (tmp)
            {
            case ENTER:
                system("cls");
                return cur;
            case UP:
                choice[cur] = 0;
                cur = (cur > 0 ? cur - 1 : menu.size() - 1);
                break;
            case DOWN:
                choice[cur] = 0;
                cur = (cur < menu.size() - 1 ? cur + 1 : 0);
                break;
            }
        }
    }
}
```

- File: [All]_Output_File.cpp: contains all methods of exporting files for saving purposes
    - The "OutCourse" exports the course information in a semester to the file where store the semester information. The information output is the name, courseID, credits, max students, and a current number of students, ...

```
void OutCourse(Course *course_head, ofstream &ofs)
{
    if (!course_head)
        return;
    while (course_head) // Output the information follows name, course ID, credits,... to file
    {
        ofs << '\n';
        ofs << course_head->Name << '\n'
            << course_head->CourseID << ' ' << course_head->Credits << ' '
            << course_head->maxStudents << ' ' << course_head->numStudents
            << ' ' << course_head->Room << '\n'
            << course_head->TeacherName
            << '\n'
            << course_head->Day << '\n'
            << course_head->Session;
        course_head = course_head->next;
    }
}
```

- "Output" exports the years, the semesters in the year, and the course information to the file 'in4smt.txt' in the corresponding folder.

```
26  void Output(Year *yearHead)
27  {
28      if (!yearHead->NoSemester)
29          return;
30
31      Semester *sem_cur = yearHead->NoSemester;
32      string out_year = to_string(yearHead->yearStart) + '_' + to_string(yearHead->yearStart + 1);
33
34      while (sem_cur)
35      {
36          // Path changes according to the current year-semester
37          string outPath = "..\\Data_file\\" + out_year + "\\smt" + to_string(sem_cur->No) + "\\in4smt.txt";
38          ofstream ofs(outPath);
39          if (!ofs)
40              return;
41          // ofs << sem_cur->No << '\n';
42          ofs << sem_cur->startDate << ' ' << sem_cur->endDate; // Output the date
43          OutCourse(sem_cur->course, ofs);
44          ofs.close();
45          Course *courseHead = sem_cur->course;
46          while (courseHead)
47          {
48              string path = createNameFile(sem_cur->Year, sem_cur->No, courseHead->Name, "score", "csv");
49              if (!checkFile(path))
50                  system(("mkdir " + path).c_str());
51              ofs.open(path);
52              outStudentCourse(courseHead->studentCourse, ofs);
53              ofs.close();
54              courseHead = courseHead->next;
55          }
56          sem_cur = sem_cur->next;
57      }
58  }
```

- "Write account" save the account and the user information to a file named 'account.csv'.

```
void WriteAccount(Account *accHead)
{
    if (!accHead)
        return;
    std::ofstream ofs("../Data_file/account.csv");
    if (!ofs)
        return;

    ofs << "Username,PassWord,Role,LastName,FirstName,Gender,SocialID,Birthday\n";
    Account *cur = accHead;
    while (cur) // follows username, pass, role, lastname, firstname, gender, social ID, birthday
    {
        ofs << cur->username << ',' << cur->password << ','
            << cur->role << ',' << cur->lastName << ','
            << cur->firstName << ',' << cur->Gender << ','
            << cur->SocialID << ',' << cur->birth;
        cur = cur->next;
        if (cur)
            ofs << "\n";
    }
    ofs.close();
}
```

- "New account" is to create a new account for a new student and link it to the account linked list.

```
//? Create an account for new student
void NewAccount(Account *accHead, string ID, Account *accTmp)
{
    Account *acc_cur = accHead;
    while (acc_cur->next)
        acc_cur = acc_cur->next;
    acc_cur->next = accTmp;
    accTmp->prev = acc_cur;
    accTmp->username = ID;          //? Set the default username to student ID
    accTmp->password = "11111111"; //? Set default password to 8 1
    return;
}
```

- "Outyear" saves the number of years in the 'years.txt' alongside classes and students in classes.

```
//? Output the year information
void Outyear(Year *yearHead)
{
    string no_year_path = "..\\Data_file\\years.txt";
    ofstream outyear(no_year_path);
    if (!outyear)
        return;

    Year *yearTMP = yearHead;
    while (yearTMP)
    {
        outyear << yearTMP->yearStart;
        Class *ClassTMP = yearTMP->Class;

        // Path changes according the the current year
        string path = "..\\Data_file\\" + to_string(yearTMP->yearStart) + '_' + to_string(yearTMP->yearStart + 1) + "\\class.txt";
        ofstream ofs(path);
        if (!ofs)
            return;
```

```
116
117         // Scan throught the list class with its students
118         while (ClassTMP)
119         {
120             ofs << ClassTMP->Name;
121             Student *StudentTMP = ClassTMP->StudentClass;
122             while (StudentTMP)
123             {
124                 ofs << "\n"
125                     << StudentTMP->ID;
126                 StudentTMP = StudentTMP->next;
127             }
128             ClassTMP = ClassTMP->next;
129             if (ClassTMP)
130                 ofs << "\n-1\n"; // Output -1 to divide out between classes
131         }
132         yearTMP = yearTMP->next;
133         if (yearTMP)
134             outyear << '\n';
135     }
136     outyear.close();
137 }
```

- "outScoreboard_StudentCourse" and "outStudentCourse" export the student's score to the corresponding csv files.

```
139 void outScoreboard_StudentCourse(ofstream &ofs, ScoreBoardCourse SBC)
140 {
141     if (SBC.totalMark < 0)
142         ofs << "X"
143             << ",";
144     else
145         ofs << SBC.totalMark << ",";
146     if (SBC.finalMark < 0)
147         ofs << "X"
148             << ",";
149     else
150         ofs << SBC.finalMark << ",";
151     if (SBC.midMark < 0)
152         ofs << "X"
153             << ",";
154     else
155         ofs << SBC.midMark << ",";
156     if (SBC.otherMark < 0)
157         ofs << "X";
158     else
159         ofs << SBC.otherMark;
160 }
```

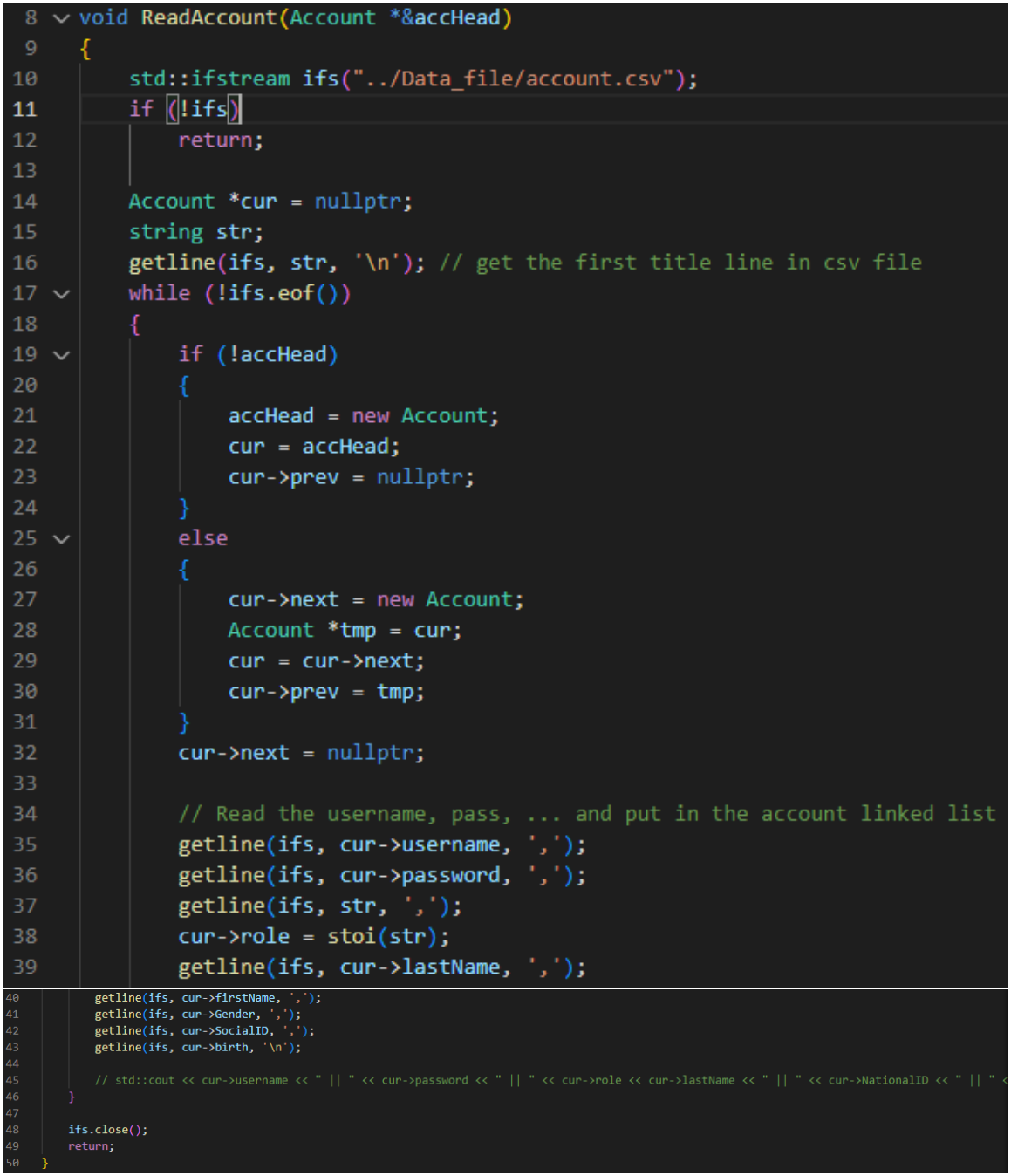
```
161  void outStudentCourse(StudentCourse *stuHead, ofstream &ofs)
162  {
163      int No = 1;
164      ofs << "No,Student ID,Fullname,Total Mark,Final Mark,Mid Mark,Other Mark";
165      while (stuHead)
166      {
167          ofs << "\n"
168              << No++ << ","
169              << stuHead->ID << ","
170              << stuHead->FullName << ",";
171          outScoreboard_StudentCourse(ofs, stuHead->ScoreBoardCourse);
172          stuHead = stuHead->next;
173      }
174  }
```

- "exportListStudentCourse" is used to create a folder and put allow those above functions to put files in it.

```
176  void exportListStudentCourse(Year *yearHead)
177  {
178      Year *ChooseYear = nullptr;
179      Semester *ChooseSem = nullptr;
180      Course *ChooseCourse = nullptr;
181      system("cls");
182      int x = 40, y = 2;
183      Render_Export(x, y);
184      x = 60, y = 15;
185      if (yearHead == nullptr)
186      {
187          Message_Warning("There are no school year.", "Error not exist");
188          // gọi lại hàm mainmenu
189          return;
190      }
191      goToXY(40, 13);
192      cout << "Select the year that contains the course you want to Export";
193      ChooseYear = chooseYearbyOption_XY(yearHead, x, y, 5);
194      cout << setw(70) << " ";
195      if (ChooseYear == nullptr)
196      {
197          // quay lại main menu
```

```
198         return;
199     }
200     if (ChooseYear->NoSemester == nullptr)
201     {
202         Message_Warning("There are no semester in this school year.", "Error not exist");
203     }
204     goToXY(40, 13);
205     cout << "Select the semester that contains the course you want to Export";
206     ChooseSem = chooseSemesterbyOption_XY(ChooseYear->NoSemester, x, y, 5);
207     cout << setw(70) << " ";
208     while (ChooseSem)
209     {
210         if (ChooseSem->course == nullptr)
211             Message_Warning("There are no course in this semester", "Error not Exist");
212         else
213         {
214             goToXY(40, 13);
215             cout << "Select the course you want to Export the scoreboard";
216             ChooseCourse = chooseCoursebyOption_XY(ChooseSem->course, x, y, 5);
217             cout << setw(70) << " ";
218             while (ChooseCourse)
219             {
220                 string path = createNameFile(ChooseSem->Year, ChooseSem->No, ChooseCourse->Name, "score", "csv");
221                 if (!checkFile(path))
222                     system(("mkdir " + path).c_str());
223                 ofstream out;
224                 out.open(path);
225                 outStudentCourse(ChooseCourse->studentCourse, out);
226                 out.close();
227                 if (!Message_YesNo("File export successful.\n Would you like to continue?", "Success"))
228                     return;
229                 ChooseCourse = chooseCoursebyOption_XY(ChooseSem->course, x, y, 5);
230             }
231         }
232         ChooseSem = chooseSemesterbyOption_XY(ChooseYear->NoSemester, x, y, 5);
233     }
234     exportListStudentCourse(yearHead);
235 }
```

- File: [All]_Readin_File.cpp: contains all methods of loading information from saved files into linked lists
  - "ReadAccount" loads back the user's accounts into the account linked list.



```
void ReadAccount(Account *&accHead)
{
    std::ifstream ifs("../Data_file/account.csv");
    if (!ifs)
        return;

    Account *cur = nullptr;
    string str;
    getline(ifs, str, '\n'); // get the first title line in csv file
    while (!ifs.eof())
    {
        if (!accHead)
        {
            accHead = new Account;
            cur = accHead;
            cur->prev = nullptr;
        }
        else
        {
            cur->next = new Account;
            Account *tmp = cur;
            cur = cur->next;
            cur->prev = tmp;
        }
        cur->next = nullptr;

        // Read the username, pass, ... and put in the account linked list
        getline(ifs, cur->username, ',');
        getline(ifs, cur->password, ',');
        getline(ifs, str, ',');
        cur->role = stoi(str);
        getline(ifs, cur->lastName, ',');
        getline(ifs, cur->firstName, ',');
        getline(ifs, cur->Gender, ',');
        getline(ifs, cur->SocialID, ',');
        getline(ifs, cur->birth, '\n');

        // std::cout << cur->username << " || " << cur->password << " || " << cur->role << cur->lastName << " || " << cur->NationalID << " || " <
    }

    ifs.close();
    return;
}
```

- "loadingFile" starts the loading process by calling the 'importYear' function.

```
53  void loadingFile(Year *&yearHead, int &numofYear)
54  {
55      cout << "Loading file" << endl;
56      importYear(yearHead, numofYear); // Read the year's stored file
57      for (int i = 0; i < 5; i++)
58      {
59          Sleep(200);
60          cout << '.'; // The effect :)))
61      }
62      cout << "\nLoading file completed." << endl;
63  }
```

- "importYear" loads the year from the file into the year linked list.

```
64  void importYear(Year *&yearHead, int &numofYear)
65  {
66      // string in_year = to_string(yearStart);
67
68      string path = "..\\Data_file\\years.txt";
69      ifstream ifs(path);
70      if (!ifs)
71      {
72          cerr << "Error: Cannot open file years.txt" << endl;
73          return;
74      }
75
76      Year *curYear = nullptr;
77      while (!ifs.eof()) // Create the year linked list according to the saved years
78      {
79          if (!yearHead)
80          {
81              yearHead = new Year;
82              curYear = yearHead;
83          }
84          else
85          {
86              Year *tmp = new Year;
87              curYear->next = tmp;
88              tmp->prev = curYear;
89              curYear = tmp;
90          }
91
92          numofYear++;
93          ifs >> curYear->yearStart;              // Get the year starting year to generate
94          importClass(curYear, curYear->yearStart); // Access the saved classes files
95      }
96  }
```

- "importClass" loads the classes and "addClass" creates the class linked list.

```
97  void addClass(Year *curYear, string ClassName)
98  {
99      if (!curYear->Class)
100     {
101         curYear->Class = new Class;
102         curYear->Class->Name = ClassName;
103     }
104     else
105     {
106         Class *curClass = curYear->Class;
107         while (curClass->next)
108             curClass = curClass->next;
109
110         curClass->next = new Class;
111         curClass->next->Name = ClassName;
112     }
113 }
```

```
114 void importClass(Year *curYear, int yearStart)
115 {
116     // access the path to the directory
117     string in_year = to_string(yearStart) + '_' + to_string(yearStart + 1);
118     string path = "..\\Data_file" + separator + in_year + separator + "class.txt";
119
120     ifstream ifs(path);
121     if (!ifs)
122     {
123         cerr << "Error: Cannot open directory for year " << yearStart << endl;
124         return;
125     }
126     // open class file
127     string classFilePath = path;
128     ifstream classFile(classFilePath);
129     if (!classFile.is_open())
130     {
131         cerr << "Error: Cannot open class file for year " << yearStart << endl;
132         return;
133     }
134
135     string line;
136     getline(classFile, line); // read the first "class" line
137     if (line.empty() || all_of(line.begin(), line.end(), [](unsigned char c)
138                                { return std::isspace(c); }))
139         return; // end of file
140
141     // read each class from the file, add to the linked list
142     Class *curClass = new Class;
143     curClass->Name = line;
144     addClass(curYear, curClass->Name);
145     curClass = curYear->Class; // add new class to linked list
```

```
146
147     Student *headStudent = nullptr;
148
149     while (getline(classFile, line))
150     {
151         if (line.empty() || all_of(line.begin(), line.end(), [](unsigned char c)
152                                    { return std::isspace(c); }))
153             break; // end of file
154
155         if (line == "-1")
156         {
157             // end of cur class, move to next class
158             getline(classFile, line); // read the class line for the next class
159             if (line.empty() || all_of(line.begin(), line.end(), [](unsigned char c)
160                                        { return std::isspace(c); }))
161                 break; // end of file
162
163             curClass = new Class;
164             curClass->Name = line;
165             addClass(curYear, curClass->Name);
166
167             Class *tmp = curYear->Class;
168             while (tmp->next)
169             {
170                 tmp = tmp->next;
171             }
172
173             curClass = tmp;
174             headStudent = nullptr;
175         }
```

```
176         else
177         {
178             if (!curClass)
179             {
180                 cerr << "Error: class not found" << endl;
181                 return;
182             }
183             // Student *curStudent = nullptr;
184             if (!headStudent)
185             {
186                 curClass->StudentClass = new Student;
187                 headStudent = curClass->StudentClass;
188                 headStudent->ID = line;
189             }
190             else
191             {
192                 Student *curStudent = new Student;
193                 curStudent->ID = line;
194                 curStudent->prev = headStudent;
195                 headStudent->next = curStudent;
196                 headStudent = curStudent;
197             }
198         }
199     }
200 
201     curClass = curYear->Class;
202     while (curClass)
203     {
204         Student *curStudent = curClass->StudentClass;
205         while (curStudent && curStudent->next)
206             curStudent = curStudent->next;
207         if (curStudent)
208             curStudent->next = nullptr;
209         curClass = curClass->next;
210     }
211 
212     classFile.close();
213     ifs.close();
214     // cout << "Class imported successfully" << endl;
215 }
```

- "Read_Sem", "Read_All_Semester" takes the 'in4smt.txt' file to trace back all the saved information of courses and semesters.

```
Semester *Read_Sem(int year, int smt)
{
    // Generate the path
    string path = "..\\Data_file\\" + to_string(year) + '_' + to_string(year + 1);
    path += "\\smt" + to_string(smt) + "\\in4smt.txt";

    ifstream ifs(path); // Open file
    if (!ifs)
        return nullptr;

    Semester *newSem = new Semester;
    newSem->Year = year;
    newSem->No = smt;                           // Get the year and semester
    ifs >> newSem->startDate >> newSem->endDate; // Get the date

    Course *newCourse = nullptr;
    while (!ifs.eof())
    {
        if (!newSem->course) // Check whether to create a new head or continue create next
        {
            newSem->course = new Course;
            newCourse = newSem->course;
        }
        else
        {
            newCourse->next = new Course;
            newCourse->next->prev = newCourse;
            newCourse = newCourse->next;
        }
        ifs.ignore();
        getline(ifs, newCourse->Name);
        ifs >> newCourse->CourseID >> newCourse->Credits >> newCourse->maxStudents >> newCourse->numStudents >> newCourse->Room;
        ifs.ignore();
        getline(ifs, newCourse->TeacherName);
        ifs >> newCourse->Day >> newCourse->Session;
    }

    ifs.close();
    return newSem;
}
```

```
Semester *Read_All_Semester(int year)
{
    Semester *semHead = nullptr;
    for (int i = 1; i <= 3; i++) // Loop to read the 3 semester in the year, return null if there isn't
    {
        Semester *newSem = Read_Sem(year, i);
        if (!newSem) // continue if null
            continue;
        if (!semHead) // If there's not, set to head
        {
            semHead = newSem;
            continue;
        }
        Semester *tmp = semHead;
        while (tmp->next)
            tmp = tmp->next;
        tmp->next = newSem;
        newSem->prev = tmp;
    }
    return semHead;
}
```

- "readScoreStudentCourse" and "readStudentCourse" takes the score of students and save them to the list.

```cpp
void readScoreStudentCourse(ScoreBoardCourse &SBC, ifstream &in)
{
    string s_totalMark, s_finalMark, s_midMark, s_otherMark;
    getline(in, s_totalMark, ',');
    getline(in, s_finalMark, ',');
    getline(in, s_midMark, ',');
    getline(in, s_otherMark, '\n');

    SBC.totalMark = SBC.finalMark = SBC.midMark = SBC.otherMark = -1;
    if (s_totalMark != "X")
        SBC.totalMark = stod(s_totalMark);
    if (s_finalMark != "X")
        SBC.finalMark = stod(s_totalMark);
    if (s_midMark != "X")
        SBC.midMark = stod(s_midMark);
    if (s_otherMark != "X")
        SBC.otherMark = stod(s_otherMark);
}
```

```cpp
void readStudentCourse(StudentCourse *&studentHead, ifstream &in)
{
    StudentCourse *cur = studentHead;
    string del = "";
    if (!in.eof())
        getline(in, del, '\n');
    while (!in.eof())
    {
        if (!cur)
            cur = studentHead = new StudentCourse;
        else
        {
            cur->next = new StudentCourse;
            cur->next->prev = cur;
            cur = cur->next;
        }
        getline(in, del, ',');
        getline(in, cur->ID, ',');
        getline(in, cur->FullName, ',');
        readScoreStudentCourse(cur->ScoreBoardCourse, in);
    }
}
```

- "readAllFileCourses" and "importScoreBoardCourse" finds the files and call the above functions to read them.

```cpp
// import data course
void readAllFileCourses(Semester *HeadSmt)
{
    ifstream in;
    Semester *curSmt = HeadSmt;
    Course *curCourse;
    while (curSmt)
    {
        curCourse = curSmt->course;
        while (curCourse)
        {
            string path = createNameFile(curSmt->Year, curSmt->No, curCourse->Name, "score", "csv");
            in.open(path);
            if (!in.is_open())
                Message_Warning(path + " is not exist.", "Error");
            else
                readStudentCourse(curCourse->studentCourse, in);
            in.close();
            curCourse = curCourse->next;
        }
        curSmt = curSmt->next;
    }
}
```

```cpp
void importScoreBoardCourse(Year *yearHead)
{
    Year *ChooseYear = nullptr;
    Semester *ChooseSem = nullptr;
    Course *ChooseCourse = nullptr;
    system("cls");
    int x = 40, y = 2;
    Render_Import(x, y);
    x = 60, y = 15;
    if (yearHead == nullptr)
    {
        Message_Warning("There are no school year.", "Error not exist");
        // gọi lại hàm mainmenu
        return;
    }
    goToXY(40, 13);
    cout << "Select the year that contains the course you want to import";
    ChooseYear = chooseYearbyOption_XY(yearHead, x, y, 5);
    cout << setw(70) << " ";
    if (ChooseYear == nullptr)
    {
        // quay lại main menu
        return;
    }
    if (ChooseYear->NoSemester == nullptr)
    {
        Message_Warning("There are no semester in this school year.", "Error not exist");
    }
    goToXY(40, 13);
    cout << "Select the semester that contains the course you want to import";
    ChooseSem = chooseSemesterbyOption_XY(ChooseYear->NoSemester, x, y, 5);
    cout << setw(70) << " ";
    while (ChooseSem)
    {
        if (ChooseSem->course == nullptr)
            Message_Warning("There are no course in this semester", "Error not Exist");
        else
        {
            goToXY(40, 13);
            cout << "Select the course you want to import the scoreboard";
            ChooseCourse = chooseCoursebyOption_XY(ChooseSem->course, x, y, 5);
            cout << setw(70) << " ";
            while (ChooseCourse)
            {
                string mess = "Put data in the  folder\nName it with format eg. Ky Nang Men : KNM.txt\nOK to continue when finish";
                Message_Warning(mess, "Notice");
                string path = createNameFile(ChooseSem->Year, ChooseSem->No, ChooseCourse->Name, "score", "csv");
                if (!checkFile(path))
                {
                    Message_Warning("The scoreboard of this course does not exist./n Please contact the teacher to get scoreboard and to try again.", " Error!");
                    break;
                }
                else
                {
                    ifstream in;
                    in.open(path);
                    readStudentCourse(ChooseCourse->studentCourse, in);
                    in.close();
                    if (!Message_YesNo("Successful import \n Would you like to continue?", "Notification"))
                        return;
                    ChooseCourse = chooseCoursebyOption_XY(ChooseSem->course, x, y, 5);
                }
            }
        }
        ChooseSem = chooseSemesterbyOption_XY(ChooseYear->NoSemester, x, y, 5);
    }
    importScoreBoardCourse(yearHead);
}
```

- File: [All]_Render_Text.cpp: storing all functions to print out the artistry of words, which functions take in value x and y to get the coordinate to print.

```
void Render_Login()
{
    TextColor(11);
    goToXY(15, 1);
    cout << " ___  ___  ___  _  _   ___  _____ _____ ___  ___ _____ _  _ _____  _____ _  _   ___  ___  ___";
    goToXY(15, 2);
    cout << " |  \\/  | / _ \\ | \\ | | / _ \\ |  _  \\|  __|| \\/ ||  __|| \\ | ||_   _| / __|\\ \\ / // __||_   _|| __|| \\/ |";
    goToXY(15, 3);
    cout << " | .  . |/ /_\\ \\|  \\| |/ /_\\ \\| | | \\/| |__  | .  . || |__  |  \\| |  | |   \\ `--.  \\ V / \\ `--.   | |  | |__  | .  . |";
    goToXY(15, 4);
    cout << " | |\\/| ||  _  || . ` ||  _  || | __ |  __| | |\\/| ||  __| | . ` |  | |    `--. \\  \\ /   `--. \\  | |  |  __| | |\\/| |";
    goToXY(15, 5);
    cout << " | |  | || | | || |\\  || | | || |_\\ \\| |___ | |  | || |___ | |\\  |  | |   /\\__/ /  | |  /\\__/ /  | |  | |___ | |  | |";
    goToXY(15, 6);
    cout << " \\_|  |_/\\_| |_/\\_| \\_/\\_/ |_/ \\____/\\____/ \\_|  |_/\\____/ \\_| \\_/  \\_/   \\____/   \\_/  \\____/   \\_/  \\____/ \\_|  |_/";

    TextColor(10);
    goToXY(60, 10);
    cout << " _     __    __ __  _  _";
    goToXY(60, 11);
    cout << "| |   / _ \\  / __|_ _|| \\| |";
    goToXY(60, 12);
    cout << "| |__| (_) || (_ || | | .` |";
    goToXY(60, 13);
    cout << "|____|\\___/  \\___|___||_|\\_|";
    TextColor(7);
}
```

```
void Render_Menu(int x, int y)
{
    goToXY(x, y++);
    TextColor(0x0A);
    cout << "___  ___ _____  _   _  _   _";
    goToXY(x, y++);
    TextColor(0x0B);
    cout << "|  \\/  ||  ___|| \\ | || | | |";
    goToXY(x, y++);
    TextColor(0x0C);
    cout << "| .  . || |__  |  \\| || | | |";
    goToXY(x, y++);
    TextColor(0x0D);
    cout << "| |\\/| ||  __| | . ` || | | |";
    goToXY(x, y++);
    TextColor(0x0E);
    cout << "| |  | || |___ | |\\  || |_| |";
    goToXY(x, y++);
    TextColor(0x0F);
    cout << "\\_|  |_/\\____/ \\_| \\_/ \\___/";
    TextColor(7);
}
```

```
58  void Render_NewInfo(int x, int y)
59  {
60      goToXY(x, y++);
61      cout << " _  _ ___ _      _   ___  _  _ ____  ___";
62      goToXY(x, y++);
63      cout << "| \\| | __\\ \\    / / |_ _|| \\| | |_  / _ \\  ";
64      goToXY(x, y++);
65      cout << "| .` | _| \\ \\/\\/ /   | | | .` | __|| (_) |";
66      goToXY(x, y++);
67      cout << "|_|\\_|___| \\_/\\_/   |___||_|\\_|_|   \\___/";
68  }
69
70  void Render_ViewCourse(int x, int y)
71  {
72      goToXY(x, y++);
73      TextColor(0x0A);
74      cout << "   _____  _____  _   _ ______  _____  _____";
75      goToXY(x, y++);
76      TextColor(0x0C);
77      cout << "  /  __ \\|  _  || | | || ___ \\/  __||  ___|";
78      goToXY(x, y++);
79      TextColor(0x0E);
80      cout << "  | |    | | | || | | || |_/ /| |__  | |__";
81      goToXY(x, y++);
82      TextColor(0x0D);
83      cout << "  | \\__/\\\\ \\_/ /| |_| || |\\ \\ /\\__/ /| |__";
84      goToXY(x, y++);
85      TextColor(0x0B);
86      cout << "   \\____/ \\___/  \\___/ \\_| \\_|\\____/ \\____/";
87      TextColor(7);
88  }
```

```
90  void Render_Account(int x, int y)
91  {
92      goToXY(x, y++);
93      TextColor(0x0A);
94      cout << "      ___  _____________  __  ___  ________";
95      goToXY(x, y++);
96      TextColor(0x0D);
97      cout << "     / _ |/ ___/ ___/ __ \\/ / / / |/ /_  __/";
98      goToXY(x, y++);
99      TextColor(0x09);
100     cout << "    / __ / /__/ /__/ /_/ / /_/ /    / / /";
101     goToXY(x, y++);
102     TextColor(0x0C);
103     cout << "   /_/ |_\\___/\\___/\\____/\\____/_/|_/ /_/";
104     TextColor(7);
105 }
106
107 void Render_NewYear(int x, int y)
108 {
109     goToXY(x, y++);
110     cout << "    _____ _____ _ _ _     __ __ _____ _____ _____";
111     goToXY(x, y++);
112     cout << "   |   | |   __| | | |   |  |  |   __|  _  | __  |";
113     goToXY(x, y++);
114     cout << "   | | | |   __| | | |   |_   _|   __|     |    -|";
115     goToXY(x, y++);
116     cout << "   |_|___|_____|_____|     |_| |_____|__|__|__|__|";
117 }
```

```
119  void Render_Semester(int x, int y)
120  {
121      goToXY(x, y++);
122      cout << "  ___ ___ __   __ ___ ___ _____ ___ ___";
123      goToXY(x, y++);
124      cout << "/ __| __|  \\/  | __/ __|_   _| __| _ \\";
125      goToXY(x, y++);
126      cout << "\\__ \\ _|| |\\/| | _|\\__ \\ | | | _||   /";
127      goToXY(x, y++);
128      cout << "|___/___|_|  |_|___|___/ |_| |___|_|_\\";
129  }
130
131  void Render_Class(int x, int y)
132  {
133      goToXY(x, y++);
134      cout << "   ______   _____           _         ______      ______  ";
135      goToXY(x, y++);
136      cout << " .' ___  | |_   _|         / \\       .' ____ \\   .' ____ \\ ";
137      goToXY(x, y++);
138      cout << "/ .'   \\_|   | |          / _ \\      | (___ \\_| | (___ \\_|";
139      goToXY(x, y++);
140      cout << "| |           | |   _     / ___ \\      _.____`.   _.____`. ";
141      goToXY(x, y++);
142      cout << "\\ `.___.'\\  _| |__/ |  _/ /   \\ \\_  | \\____) | | \\____) |";
143      goToXY(x, y++);
144      cout << " `.____ .' |________| |____| |____|  \\______.'  \\______.";
145  }
```

```
146  void Render_Course(int x, int y)
147  {
148      goToXY(x, y++);
149      cout << "   _____     __    ____  ____  ______     _____    _______";
150      goToXY(x, y++);
151      cout << " .' ___  |  .'   `. |_   _||_  _||_   __ \\   .' ___ \\  |_   __ |";
152      goToXY(x, y++);
153      cout << "/ .'   \\_| /  .-.  \\  | |    | |    | |__) |  | (__ \\_|   | |_ \\_|";
154      goToXY(x, y++);
155      cout << "| |        | |   | |  | '    ' |    |  __ /    _.____`.    |  _| _  ";
156      goToXY(x, y++);
157      cout << "\\ `.___.'  \\  `-'  /   \\ \\__/ /     _| |  \\ \\_ | \\____) |  _| |__/ |";
158      goToXY(x, y++);
159      cout << " `.____ .'  `.___.'     `.__.'     |____| |___| \\_____.' |_______|";
160  }
161
162  void Render_Student(int x, int y)
163  {
164      goToXY(x, y++);
165      cout << "  _____   ________  ____  ____  _____    _______  ___  ____  ________       ";
166      goToXY(x, y++);
167      cout << ".' ___ \\  |  _   _  | |_  _||_  _| |_  _ `. |_  __ | |_  \\|_  _| | _  _ |     ";
168      goToXY(x, y++);
169      cout << "| (__ \\_| |_/ | | \\_|  | |    | |    | | `. \\  | |_ \\_|   |  \\ | |   |_/ | | \\_| ";
170      goToXY(x, y++);
171      cout << " _.____`.      | |      | '    ' |    | |  | |   | _| _    | |\\ \\| |      | |       ";
172      goToXY(x, y++);
173      cout << "| \\____) |    _| |_      \\ \\__/ /     _| |_.' /  _| |__/ |  _| |_\\ |_    _| |_     ";
174      goToXY(x, y++);
175      cout << " \\_____.'   |_____|      `.__.'     |_____.' |________| |_____|\\____|   |_____|     ";
176  }
```

```
178    void Render_View(int x, int y)
179    {
180        goToXY(x, y++);
181        cout << "  ____    ____    _____    _________    ____        ____  ";
182        goToXY(x, y++);
183        cout << "|_   _| |_   _| |_   _|  |_   __   | |_  _|     |_  _|";
184        goToXY(x, y++);
185        cout << "  \\ \\   / /      | |      | |_ \\_|    \\ \\  /\\  / /  ";
186        goToXY(x, y++);
187        cout << "   \\ \\ / /       | |      |  _| _      \\ \\/  \\/ /";
188        goToXY(x, y++);
189        cout << "    \\ ' /       _| |_    _| |__/ |      \\  /\\  / ";
190        goToXY(x, y++);
191        cout << "     \\_/       |_____| |_________|       \\/  \\ ";
192    }
193
194    void Render_StudentCourse()
195    {
196        int x = 30, y = 1;
197        Render_Student(x, y);
198        y = 7;
199        x = 40;
200        Render_Course(x, y);
201    }
```

```
203    void Render_ScoreBoard(int x, int y)
204    {
205        goToXY(x, y++);
206        cout << "   _____      _____     __     ______      _______  _____       __          _       ______      _____          ";
207        goToXY(x, y++);
208        cout << " .' ___ \\   .' __  |  .'   `.  |_   __ \\    |_   __ | |_  _ \\    .'   `.      / \\     |_   __ \\    |_   _ `.    ";
209        goToXY(x, y++);
210        cout << " | (__ \\_| / .'  \\_| /  .--.  \\   | |__) |     | |_ \\_|   | |_) |  /  .--.  \\    / _ \\      | |__) |     | | `. \\    ";
211        goToXY(x, y++);
212        cout << "  _.___`.  | |       | |   | |   |  __ /      |  _| _    |  __'.  | |   | |    / ___ \\     |  __ /      | |  | |   ";
213        goToXY(x, y++);
214        cout << " | \\____) | \\ `.___.'\\\\ \\\\  `-'  /  _| |  \\\\ \\\\_   _| |__/ |  _| |_) | \\\\  `-'  /  _/ /   \\\\ \\\\_   _| |  \\\\ \\\\_   _| |_.' /    ";
215        goToXY(x, y++);
216        cout << "  \\\\_____.'  `.___ .'  `.__.'  |____| |____| |_______| |_____/   `.__.'  |____| |____| |____| |____| |_____.'       ";
217    }
218    void Render_Update(int x, int y)
219    {
220        goToXY(x, y++);
221        cout << "  ____  ____   _____    _____         _       _______   _______  ";
222        goToXY(x, y++);
223        cout << " |_   _||_   _| |_   __ \\  |_   _ `.      / \\     |  _   _  | |_   __  | ";
224        goToXY(x, y++);
225        cout << "   | |    | |     | |__) |   | | `. \\    / _ \\    |_/ | | \\_|   | |_ \\_| ";
226        goToXY(x, y++);
227        cout << "   | '    ' |     |  __/    | |  | |   / ___ \\       | |       | _| _  ";
228        goToXY(x, y++);
229        cout << "    \\ \\__/ /     _| |_     _| |_.' /  _/ /   \\ \\_    _| |_     _| |__/ | ";
230        goToXY(x, y++);
231        cout << "     `.__.'     |_____|   |_____.' |____| |____|  |_____|  |_______| ";
232    }
```

```
233  void Render_Export(int x, int y)
234  {
235      goToXY(x, y++);
236      cout << "  ________   ___  ___   _______      __     _______      ________  ";
237      goToXY(x, y++);
238      cout << " |_   __  | |_  _||_  _| |_   __ \\   .'   `.  |_   __ \\    | _   _  | ";
239      goToXY(x, y++);
240      cout << "   | |_ \\_|   \\ \\  / /     | |__) | /  .-.  \\   | |__) |   |_/ | | \\_| ";
241      goToXY(x, y++);
242      cout << "   |  _| _     > `' <      |  __/  | |   | |   |  __ /        | |     ";
243      goToXY(x, y++);
244      cout << "  _| |__/ |  _/ /'`\\ \\_   _| |_     \\  `-'  /  _| |  \\ \\_     _| |_    ";
245      goToXY(x, y++);
246      cout << " |________| |____||____| |_____|     `.___.'  |____| |___|   |_____|   ";
247  }
248  void Render_Import(int x, int y)
249  {
250      goToXY(x, y++);
251      cout << "  _____   ___    ___   _______      __     _______      ________  ";
252      goToXY(x, y++);
253      cout << " |_   _| |_   \\  /   _| |_   __ \\   .'   `.  |_   __ \\    | _   _  | ";
254      goToXY(x, y++);
255      cout << "   | |     |   \\/   |     | |__) | /  .-.  \\   | |__) |   |_/ | | \\_| ";
256      goToXY(x, y++);
257      cout << "   | |     | |\\  /| |     |  __/  | |   | |   |  __ /        | |     ";
258      goToXY(x, y++);
259      cout << "  _| |_   _| |_\\/_| |_   _| |_     \\  `-'  /  _| |  \\ \\_     _| |_    ";
260      goToXY(x, y++);
261      cout << " |_____| |_____||_____| |_____|     `.___.'  |____| |___|   |_____|   ";
262  }
```

```
263  void Render_ScoreBoardCourse()
264  {
265      int x = 25, y = 1;
266      Render_ScoreBoard(x, y);
267      y = 7;
268      x = 45;
269      Render_Course(x, y);
270  }
271
272  void Render_ScoreBoardClass()
273  {
274      int x = 20, y = 1;
275      Render_ScoreBoard(x, y);
276      x = 45;
277      y = 7;
278      Render_Class(x, y);
279  }
280
281  void Render_StudentClass()
282  {
283      int x = 30, y = 1;
284      Render_Student(x, y);
285      x = 42;
286      y = 7;
287      Render_Class(x, y);
288  }
```

- File: [All]_Role_Menu.cpp: The menu of actions an account has.
  - "Staff_Main" is the function, the menu of action for a staff showing when enter, which is account options, view, add, modify information, import or export data.

```cpp
void Staff_Main(Year *yearHead, Account *accHead, string &user, string &pass, int &role)
{
    system("cls");
    Render_Menu(59, 3);

    vector<string> menu;
    menu.push_back("Account");
    menu.push_back("View");   //! 1 nui` view j do
    menu.push_back("Add");    //! 1 nui` add j do
    menu.push_back("Modify"); //! 1 nui` modify j do
    menu.push_back("Import / Export");
    menu.push_back("Quit");

    int choice = Draw_XY(menu, 60, 12, 7, 25);
    int ye;
    Year *year_cur;
    string mess;
    Student *student_cur = nullptr;
    Semester *curSem;
    switch (choice) // Do the choice
    {
    case 0: //? Account
        if (AccountAlteration(yearHead, accHead, user, pass, role))
            return;
        if (role == 1)
        {
            student_cur = findStudentbyID(user, yearHead);
            Student_Main(yearHead, accHead, student_cur, user, pass, role);
            return;
        }
        break;
    case 1:
        //? View
        Staff_View(yearHead);
        break;
    case 2:
        //? Adding year/semester/course
        New_Stuff(yearHead, accHead); // case 2:
        SyncFullName(yearHead, accHead);
        break;
    case 3:
        //? Modify year/semester/course
        system("cls");
        initModify(yearHead);
        SyncFullName(yearHead, accHead);
        // Recursion back to the StaffMain function
        break;
    case 4:
        //? Import/ Export
        Interface_Import_Export(yearHead);
        break;
    case 5:
        //? Save changes and quit
        mess = "Do you want to save all the changes?";
        if (Message_YesNo(mess, "Notice!"))
        {
            Output(yearHead);  // write down all the year
            Outyear(yearHead); // write down each year in4
            mess = "File save!\nNow clean up and close!";
            Message_Warning(mess, "Quit");
        }

        cout << "Cleaned up the system";
        Pause();
        return;
    }
    Staff_Main(yearHead, accHead, user, pass, role);
    return;
}
```

```
77  void Teacher_Main()
78  {
79      system("cls");
80      Render_Menu(59, 3);
81
82      vector<string> menu;
83      menu.push_back("Account");
84      //? push command here
85
86      int choice = Draw_XY(menu, 66, 12, 5, 20);
87      int ye;
88      Year *year_cur;
89      string mess;
90      // todo Teacher co the xem khoa, xem hoc sinh,...
91      switch (choice) // Do the choice
92      {
93      case 0:
94          //? Account options
95          // AccountAlteration();
96          break;
97      case 1:
98          //?
99          // ViewCourse();
100         break;
101     case 2:
102         //? Adding year/semester/course
103         // New_Stuff();
104         break;
105     case 3:
106         //? Modify year/semester/course
107         // initModify();
108         break;
109     case 4:
110         //? Save changes and quit
111
112         break;
113     }
114 }
```

- "Student_Main" is the menu options for student accounts that students can manage the account, view the course, and score.

```
116  void Student_Main(Year *yearHead, Account *accHead, Student *student_cur, string &user, string &pass, int &role)
117  {
118      system("cls");
119      Render_Student(35, 3);
120
121      vector<string> menu;
122      menu.push_back("Account");
123      menu.push_back("Your Course");
124      menu.push_back("Your ScoreBoard");
125      menu.push_back("Quit");
126
127      int opt = Draw_XY(menu, 60, 12, 4, 24);
128      switch (opt)
129      {
130      case 0:
131          //?
132          if (AccountAlteration(yearHead, accHead, user, pass, role))
133              return;
134          if (role == 3 || role == 2)
135          {
136              Staff_Main(yearHead, accHead, user, pass, role);
137              return;
138          }
139          break;
140      case 1:
141          //? Add a new semester
142          Interface_ViewCoursesOfUser(student_cur, yearHead);
143          // Recursion back to the StaffMain function
144          break;
145      case 2:
146          //? Add a new course
147          Interface_ViewScoreBoardOfUser(student_cur, yearHead);
148          // Recursion back to the StaffMain function
149          break;
150      case 3:
151          return;
152      }
153      Student_Main(yearHead, accHead, student_cur, user, pass, role);
154  }
```

- "Main_Menu" is called when the user enters the system after login successfully. It will decide based on the account role and show the corresponding menu.

```
156  void Main_Menu(Year *yearHead, Account *accHead, string &user, string &pass, int &role)
157  {
158      system("cls");
159      if (role == 2)
160          role = 3;
161      Student *student_cur = nullptr;
162
163      //! Read year
164      //! In add the semester in each read year
165      // SyncFullName(yearHead, accHead);
166      switch (role)
167      {
168      case 1:
169          //? StudentMain
170          student_cur = findStudentbyID(user, yearHead);
171          Student_Main(yearHead, accHead, student_cur, user, pass, role);
172          break;
173      case 3:
174          //? StaffMain
175          Staff_Main(yearHead, accHead, user, pass, role);
176          break;
177      }
178
179      return;
180  }
```

- File: [All]_Sort.cpp: has the function to swap the year
  - "SwapYear" is used to swap the inside information that stores in a year linked list.

```
3   void SwapYear(Year *year1, Year *year2)
4   {
5       year1->yearStart = year1->yearStart + year2->yearStart;
6       year2->yearStart = year1->yearStart - year2->yearStart;
7       year1->yearStart = year1->yearStart - year2->yearStart;
8
9       Class *tmp_Class = year1->Class;
10      year1->Class = year2->Class;
11      year2->Class = tmp_Class;
12
13      Semester *tmp_Sem = year1->NoSemester;
14      year1->NoSemester = year1->NoSemester;
15      year2->NoSemester = tmp_Sem;
16  }
```

  - "SortYear_Descending" swaps the year into the correct order.

```
17  Year *SortYear_Descending(Year *yearHead)
18  {
19      Year *year_i;
20      Year *year_j;
21      for (year_i = yearHead; year_i && year_i->next != nullptr; year_i = year_i->next)
22          for (year_j = year_i->next; year_j != nullptr; year_j = year_j->next)
23              if (year_i->yearStart < year_j->yearStart)
24                  SwapYear(year_i, year_j);
25      while (year_i && year_i->prev)
26          year_i = year_i->prev;
27      return year_i;
28  }
```

- File: [All]_Sync.cpp: has functions that allow the program to sync information across linked lists
  - "SyncInForStudent" traverses the linked list of years and finds the correct account node to link to the student.

```
void SyncInForStudent(Year *yearHead, Account *accHead)
{
    Year *year_cur;
    Class *class_cur;
    Student *student_cur;
    Account *account_cur;
    year_cur = yearHead;
    while (year_cur)
    {
        class_cur = year_cur->Class;
        while (class_cur)
        {
            student_cur = class_cur->StudentClass;
            while (student_cur)
            {
                account_cur = accHead;
                while (account_cur && student_cur->ID != account_cur->username)
                    account_cur = account_cur->next;
                if (account_cur == nullptr)
                {
                    Message_Warning("Student information with ID: " + student_cur->ID + " does not exist.", "error import");
                }
                else
                {
                    student_cur->accStudent = account_cur;
                }
                student_cur = student_cur->next;
            }
            class_cur = class_cur->next;
        }
        year_cur = year_cur->next;
    }
}
```

- "SyncCourse" gets all the course that student has put inside a linked list for better management.

```
void SyncCourse(Year *yearHead)
{
    Year *year_cur1 = yearHead;
    Year *year_cur2;
    while (year_cur1)
    {
        Semester *sem_cur = year_cur1->NoSemester;
        while (sem_cur)
        {
            Course *cse = sem_cur->course;
            while (cse)
            {
                StudentCourse *stcse = cse->studentCourse;
                while (stcse)
                {
                    year_cur2 = yearHead;
                    while (year_cur2)
                    {
                        Class *class_cur = year_cur2->Class;
                        while (class_cur)
                        {
                            Student *stclass = class_cur->StudentClass;
                            while (stclass && stclass->ID != stcse->ID)
                                stclass = stclass->next;
                            if (stclass != nullptr)
                            {
                                if (stclass->course == nullptr)
                                    stclass->course = new CourseStudent;
                                else
                                {
                                    CourseStudent *tmp = new CourseStudent;
                                    tmp->next = stclass->course;
                                    stclass->course = tmp;
                                }
                                stclass->course->course = cse;
                                break;
                            }
                            class_cur = class_cur->next;
                        }
                        year_cur2 = year_cur2->next;
                    }
                    stcse = stcse->next;
                }
                cse = cse->next;
            }
            sem_cur = sem_cur->next;
        }
        year_cur1 = year_cur1->next;
    }
}
```

- "SyncFullName" traverse the year and account linked list to get the name of the student corresponding to the student ID.

```
93  //? get the fullname from account linked list
94  void SyncFullName(Year *yearHead, Account *accHead)
95  {
96      Year *year_cur = yearHead;
97      while (year_cur)
98      {
99          Semester *sem_cur = year_cur->NoSemester;
100         while (sem_cur)
101         {
102             Course *cse = sem_cur->course;
103             while (cse)
104             {
105                 StudentCourse *stcse = cse->studentCourse;
106                 while (stcse)
107                 {
108                     Account *acc_cur = accHead;
109                     while (acc_cur)
110                     {
111                         if (stcse->FullName == "" && acc_cur->username == stcse->ID)
112                         {
113                             stcse->FullName = acc_cur->lastName + ' ' + acc_cur->firstName;
114                             break;
115                         }
116                         acc_cur = acc_cur->next;
117                     }
118                     stcse = stcse->next;
119                 }
120                 cse = cse->next;
121             }
122             sem_cur = sem_cur->next;
123         }
124         year_cur = year_cur->next;
125     }
126 }
```

- File: [All]_Utilities.cpp: stores all the functions that use across files for managing the cursor and color purpose
  - "ShowConsoleCursor" is the function that shows or hides the cursor.

```
3   //? Show the blinking cursor in the terminal, disable when using arrow
4   void ShowConsoleCursor(bool visible)
5   {
6       HANDLE console = GetStdHandle(STD_OUTPUT_HANDLE);
7       CONSOLE_CURSOR_INFO lpCursor;
8       lpCursor.bVisible = visible;
9       lpCursor.dwSize = 20;
10      SetConsoleCursorInfo(console, &lpCursor);
11  }
```

  - "Pause": use the ‘conio.h’ style to replicate the 'pause' command line function.

```
12  void Pause()
13  {
14      while (1)
15          if (_kbhit())
16          {
17              int tmp = _getch();
18              break;
19          }
20      return;
21  }
```

- "Message_Warning" and "Message_YesNo" are functions to show a message box with the inputting text and a button for users to click.

```
22  //? Popup message box
23  void Message_Warning(string message, string title)
24  {
25      EnableWindow(GetConsoleWindow(), false);
26      MessageBoxA(NULL, message.c_str(), title.c_str(), MB_OK | MB_ICONINFORMATION);
27      EnableWindow(GetConsoleWindow(), true);
28  }
29
30  bool Message_YesNo(string message, string title)
31  {
32      EnableWindow(GetConsoleWindow(), false);
33      int result = MessageBoxA(NULL, message.c_str(), title.c_str(), MB_YESNO | MB_ICONQUESTION);
34      EnableWindow(GetConsoleWindow(), true);
35      return (result == IDYES ? true : false);
36  }
```

- "TextColor" changes the color of the console text.

```
38  //? Change the text color in terminal
39  void TextColor(int x) // X là mã màu
40  {
41      // Các hàm này là hàm thao tác API với windows bạn cứ coppy thôi không cần phải hiểu quá sâu
42      SetConsoleTextAttribute(GetStdHandle(STD_OUTPUT_HANDLE), x);
43  }
```

- "goToXY" moves the cursor to the inputting position.

```
45  //? Move the cursor to the position (x, y) in terminal
46  void goToXY(int x, int y)
47  {
48      HANDLE console = GetStdHandle(STD_OUTPUT_HANDLE);
49      COORD cursorPosition;
50      cursorPosition.X = x;
51      cursorPosition.Y = y;
52      SetConsoleCursorPosition(console, cursorPosition);
53  }
```

- "Limit_Input" replicate the std::cin by using ‘conio.h’ library and limiting the inputting length.

```cpp
//? Limit the input string to a certain length and without some special characters and arrow keys
string Limit_Input(int x, int y, int limit, int color)
{
    ShowConsoleCursor(true);
    goToXY(x, y);
    TextColor(color);
    string input;
    char c;
    int i = 0;
    while (true)
    {
        c = _getch();
        if (c == 13 && i > 0)
            break;
        else if (c == 8)
        {
            if (i > 0)
            {
                i--;
                input.pop_back();
                goToXY(x + i, y);
                cout << " ";
                goToXY(x + i, y);
            }
        }
        else if (c == 23)
            while (i > 0)
            {
                i--;
                input.pop_back();
                goToXY(x + i, y);
                cout << " ";
                goToXY(x + i, y);
            }
        else if (c == -32)
        {
            c = _getch();
            continue;
        }
        //? 33 <= c <= 39: ! " # $ % & '
        //? 42 <= c <= 57:* + , - . / 0 -> 9
        //? 63 <= c <= 90: ? @ A -> Z
        //? 94: ^
        //? 97 <= c <= 122: a -> z
        else if (i < limit && ((c >= 63 && c <= 90) || (c >= 97 && c <= 122) || (c >= 42 && c <= 57) || (c >= 33 && c <= 39) || c == 42 || c == 94))
        {
            input.push_back(c);
            cout << c;
            i++;
        }
    }
    TextColor(7);
    ShowConsoleCursor(false);
    return input;
}
```

- "resizeWindow" resizes the console window to the fixed size.

```cpp
void resizeWindow(int x, int y)
{
    HWND console = GetConsoleWindow();
    RECT r;
    GetWindowRect(console, &r);
    MoveWindow(console, r.left, r.top, x, y, TRUE);
}
```

- "hideScrollBar" hides the console's scroll bar.

```
119  void hideScrollBar()
120  {
121      HANDLE console = GetStdHandle(STD_OUTPUT_HANDLE);
122      CONSOLE_SCREEN_BUFFER_INFO screenBufferInfo;
123      GetConsoleScreenBufferInfo(console, &screenBufferInfo);
124
125      COORD new_screen_buffer_size;
126
127      new_screen_buffer_size.X = screenBufferInfo.srWindow.Right - screenBufferInfo.srWindow.Left + 1;
128      new_screen_buffer_size.Y = screenBufferInfo.srWindow.Bottom - screenBufferInfo.srWindow.Top + 1;
129
130      SetConsoleScreenBufferSize(console, new_screen_buffer_size);
131  }
```

- "initWindow" initializes the console window by setting the size, and hiding the scroll bar,...

```
133  void initWindow(int width, int length)
134  {
135      resizeWindow(width, length);
136      hideScrollBar();
137  }
```

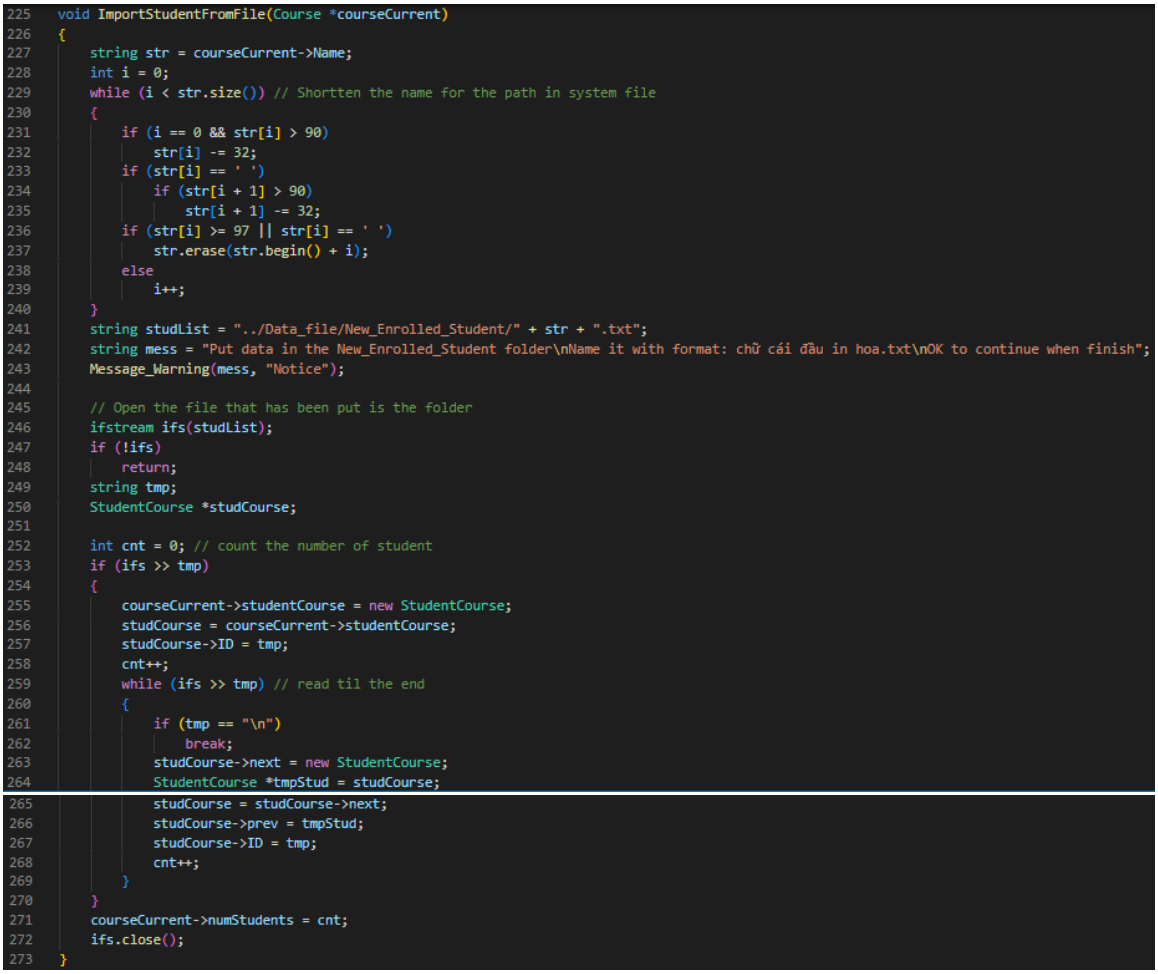
ii. [Sem] Section

- File: [Sem]_Add_Course_Student.cpp: holds functions that allow users to create new semesters or courses with students.
  - "AddNewCourse" takes in the current chosen semester and the year linked list, then it prints to the console the necessary information for the user to input including the course ID, its name, teacher, its credits, the maximum number of students, the room and its time table. Finally, the function creates a new node at the end of the course inside the selected semester and returns back the node itself.

```
122 Course *AddNewCourse(Semester *semCurrent, Year *yearHead)
123 {
124     system("cls");
125     Render_NewInfo(50, 3);
126 
127     TextColor(14);
128     goToXY(52, 9);
129     cout << "Course ID";
130     goToXY(52, 14);
131     cout << "Course name";
132     goToXY(52, 19);
133     cout << "Teacher name";
134 
135     goToXY(51, 25);
136     cout << "Credit";
137     goToXY(73, 25);
138     cout << "Max student";
139     goToXY(51, 29);
140     cout << "Room";
141     goToXY(66, 29);
142     cout << "Day";
143     goToXY(80, 29);
144     cout << "Session";
145 
146     TextColor(63);
147     for (int i = 0; i < 3; i++)
148     {
149         for (int j = 0; j < 3; j++)
150         {
151             goToXY(50, 10 + 5 * i + j);
152             cout << "                                                    ";
153         }
154         goToXY(59, 24 + i);
155         cout << "          "; // Credit
156         goToXY(86, 24 + i);
157         cout << "          "; // Max stud
158 
159         goToXY(57, 28 + i);
160         cout << "       "; // Room
```

```
161         goToXY(71, 28 + i);
162         cout << "       "; // Day
163         goToXY(89, 28 + i);
164         cout << "       "; // Session
165     }
166 
167     string id = Limit_Input(52, 11, 10, 63);
168     Course *courseCurrent, *prev = nullptr;
169     if (!semCurrent->course) // Generate the head if there is not
170     {
171         semCurrent->course = new Course;
172         courseCurrent = semCurrent->course;
173     }
174     else
175     {
176         courseCurrent = semCurrent->course;
177         while (courseCurrent->next)
178         {
179             if (courseCurrent->CourseID == id)
180             {
181                 // ask if they want to change its in4
182                 string mess = "This course already exists, do you want to modify it?";
183                 if (Message_YesNo(mess, "Notice"))
184                     // Modify the semester
185                     modifyCourse(yearHead, semCurrent->Year);
186                 return nullptr;
187             }
188             courseCurrent = courseCurrent->next;
189         }
190         // create a new node at last pos
191         prev = courseCurrent;
192         courseCurrent->next = new Course;
193         courseCurrent = courseCurrent->next;
194         courseCurrent->prev = prev;
195     }
196 
197     courseCurrent->CourseID = id;
198 
199     courseCurrent->Name = Limit_Input(52, 16, 30, 63);
200     courseCurrent->TeacherName = Limit_Input(52, 21, 30, 63);
201     courseCurrent->Credits = stoi(Limit_Input(64, 25, 1, 63));
202     courseCurrent->maxStudents = stoi(Limit_Input(89, 25, 3, 63));
203     courseCurrent->Room = Limit_Input(59, 29, 3, 63);
204     courseCurrent->Day = Limit_Input(73, 29, 3, 63);
205     while (!isDay(courseCurrent->Day))
206     {
207         Message_Warning("Invalid day format!\n(MON/TUE/WED/THU/FRI/SAT/SUN)", "Error");
208         goToXY(71, 29);
209         cout << "       ";
210         courseCurrent->Day = Limit_Input(73, 29, 3, 63);
211     }
212 
213     courseCurrent->Session = Limit_Input(91, 29, 2, 63);
214     while (!isSession(courseCurrent->Session))
215     {
216         Message_Warning("Invalid session format!\n(S1/S2/S3/S4)", "Error");
217         goToXY(89, 29);
218         cout << "       ";
219         courseCurrent->Session = Limit_Input(91, 29, 2, 63);
220     }
221     return courseCurrent;
222 }
```

- "ImportStudentFromFile" makes the user input the file name that the user put in the specific folder, then it reads the file and imports the student information into the student linked list.



```
void ImportStudentFromFile(Course *courseCurrent)
{
    string str = courseCurrent->Name;
    int i = 0;
    while (i < str.size()) // Shortten the name for the path in system file
    {
        if (i == 0 && str[i] > 90)
            str[i] -= 32;
        if (str[i] == ' ')
            if (str[i + 1] > 90)
                str[i + 1] -= 32;
        if (str[i] >= 97 || str[i] == ' ')
            str.erase(str.begin() + i);
        else
            i++;
    }
    string studList = "../Data_file/New_Enrolled_Student/" + str + ".txt";
    string mess = "Put data in the New_Enrolled_Student folder\nName it with format: chữ cái đầu in hoa.txt\nOK to continue when finish";
    Message_Warning(mess, "Notice");

    // Open the file that has been put is the folder
    ifstream ifs(studList);
    if (!ifs)
        return;
    string tmp;
    StudentCourse *studCourse;

    int cnt = 0; // count the number of student
    if (ifs >> tmp)
    {
        courseCurrent->studentCourse = new StudentCourse;
        studCourse = courseCurrent->studentCourse;
        studCourse->ID = tmp;
        cnt++;
        while (ifs >> tmp) // read til the end
        {
            if (tmp == "\n")
                break;
            studCourse->next = new StudentCourse;
            StudentCourse *tmpStud = studCourse;
            studCourse = studCourse->next;
            studCourse->prev = tmpStud;
            studCourse->ID = tmp;
            cnt++;
        }
    }
    courseCurrent->numStudents = cnt;
    ifs.close();
}
```

- "AddStudentByHand" allows the user to input each student's information, then creates a new student node at the end of the linked list.

```
276  void AddStudentByHand(Course *courseCurrent)
277  {
278      // cout << "Enter the student ID one by one! (Enter -1 to stop)\n\n";
279
280      StudentCourse *studCourse = courseCurrent->studentCourse;
281      goToXY(60, 17);
282      cout << "Enter student ID (-1 to stop)";
283      int cnt = 0, i = 0;
284      string line;
285      while (1)
286      {
287          TextColor(63);
288          goToXY(58, 19 + i);
289          cout << "                              ";
290          goToXY(65, 19 + i);
291          line = Limit_Input(65, 19 + i, 8, 63);
292
293          if (line == "-1")
294              break;
295          if (line.size() != 8)
296          {
297              if (Message_YesNo("Invalid ID format!", "Error"))
298                  continue;
299              else
300                  break;
301          }
302          i++;
303          if (!studCourse)
304          {
305              studCourse = new StudentCourse;
306              courseCurrent->studentCourse = studCourse;
307          }
308          else
309          {
310              studCourse->next = new StudentCourse;
311              studCourse->next->prev = studCourse;
312              studCourse = studCourse->next;
313          }
314          studCourse->ID = line;
315          cnt++;
316      }
317      courseCurrent->numStudents = cnt;
318
319      cout << "\nDone adding new student!\n";
320      Message_Warning("Done adding new student to course!", "Success");
321      return;
322  }
```

- "AddingCourse" gives the user options to choose the way they want to add a student to the course, either by importing from a file or by hand.

```cpp
void AddingCourse(Semester *semCurrent, Year *yearHead)
{
    if (!semCurrent)
        return;
    Course *courseCurrent = AddNewCourse(semCurrent, yearHead);

    vector<string> menu;
    system("cls");
    Render_Class(50, 2);
    //! add a title
    menu.push_back("Import student from file");
    menu.push_back("Add student by hand");
    menu.push_back("Back");

    int choice = Draw_XY(menu, 55, 12, 3, 30, 63);

    switch (choice) // Access according to the choice
    {
    case 0:
        Message_Warning("Importing student from file!", "Notice");
        ImportStudentFromFile(courseCurrent);
        break;
    case 1:
        Message_Warning("Adding student by hand!", "Notice");
        AddStudentByHand(courseCurrent);
        break;
    case 2:
        return;
    }

    string mess = "Do you want to add another course to this semester?";
    if (Message_YesNo(mess, "Notice"))
        AddingCourse(semCurrent, yearHead);
    return;
}
```

- File: [Sem]_Add_Semester.cpp: holds funtions that allow users to create new semesters or courses with students.
  - “compareDate”: the function takes in 2 date strings and compare them, return -1, 0, 1 accordingly.

```
int compareDate(string date1, string date2)
{
    // Extract day, month, and year from date1
    int day1 = std::stoi(date1.substr(0, 2));
    int month1 = std::stoi(date1.substr(3, 2));
    int year1 = std::stoi(date1.substr(6, 4));

    // Extract day, month, and year from date2
    int day2 = std::stoi(date2.substr(0, 2));
    int month2 = std::stoi(date2.substr(3, 2));
    int year2 = std::stoi(date2.substr(6, 4));

    // Compare the years
    if (year1 < year2)
        return -1;
    else if (year1 > year2)
        return 1;

    // Compare the months
    if (month1 < month2)
        return -1;
    else if (month1 > month2)
        return 1;

    // Compare the days
    if (day1 < day2)
        return -1;
    else if (day1 > day2)
        return 1;

    // The dates are equal
    return 0;
}
```

- "AddSemester" is a function that takes in the year linked list, creates a new semester node at the end of the selected year, then allows the user to input the starting and ending date of the semester. Finally, it returns the semester that just has been created.

```
Semester *AddSemester(Year *yearHead)
{
    system("cls");
    Render_Semester(50, 3);
    // Print the list of year
    Year *year_cur = yearHead;
    // Get the year and semester
    year_cur = chooseYearbyOption_XY(yearHead, 60, 12, 5);
    if (!year_cur)
        return nullptr;
    string tmp;

    vector<string> small_menu;
    small_menu.push_back("1st semester");
    small_menu.push_back("2nd semester");
    small_menu.push_back("3rd semester");
    small_menu.push_back("Back");
    int N = 0;
    N = Draw_XY(small_menu, 60, 12, 4, 20, 63) + 1;
    if (N == 4)
    {
        AddSemester(yearHead);
        return nullptr;
    }

    Semester *sem_cur = year_cur->NoSemester;
    Semester *prev = sem_cur;
    while (sem_cur)
    {
        if (sem_cur->No == N) // If already had, ask to modify
        {
            string mess = "This semester already exists!";
            Message_Warning(mess, "Notice");
            return AddSemester(yearHead);
        }
        prev = sem_cur;
        if (!sem_cur->next)
            break;
        sem_cur = sem_cur->next;
```

```
        sem_cur = sem_cur->next;
    }

    if (!prev) // if there is no semester to begin with, create a new head
    {
        year_cur->NoSemester = new Semester;
        sem_cur = year_cur->NoSemester;
    }
    else // create a new semester at the last position
    {
        prev->next = new Semester;
        sem_cur = prev->next;
        sem_cur->prev = prev;
    }

    sem_cur->No = N;
    sem_cur->Year = year_cur->yearStart;

    system("cls");
    string date1, date2;

    Render_Semester(50, 3);
    TextColor(63);
    for (int i = 0; i < 3; i++)
    {
        goToXY(50, 13 + i);
        cout << "                                                ";
    }
    goToXY(52, 14);
    cout << "Starting date (dd/mm/yyyy): "; // Get the date and ensure its format

    tmp = Limit_Input(80, 14, 10, 63);

    while (!isValidDate(tmp))
    {
        if (tmp == "-1")
        {
            if (sem_cur->prev)
                sem_cur->prev->next = nullptr;
```

```
            else
                year_cur->NoSemester = nullptr;
            delete sem_cur;
            return nullptr;
        }
        Message_Warning("Invalid date format!\nEnter again!", "Error");
        goToXY(50, 14);
        TextColor(63);
        cout << "                                                  ";
        goToXY(52, 14);
        cout << "Starting date (dd/mm/yyyy): "; // Get the date and ensure its format
        tmp = Limit_Input(80, 14, 10, 63);
    }
    // sem_cur->startDate = tmp;
    date1 = tmp;

    TextColor(63);
    for (int i = 0; i < 3; i++)
    {
        goToXY(50, 17 + i);
        cout << "                                                  ";
    }
    goToXY(52, 18);
    cout << "Ending date (dd/mm/yyyy): ";
    tmp = Limit_Input(80, 18, 10, 63);

    while (!isValidDate(tmp) || compareDate(date1, tmp) != -1)
    {
        if (tmp == "-1")
        {
            if (sem_cur->prev)
                sem_cur->prev->next = nullptr;
            else
                year_cur->NoSemester = nullptr;
            delete sem_cur;
            return nullptr;
        }

        if (isValidDate(tmp) && compareDate(date1, tmp) != -1)
            Message_Warning("Invalid date!\nEnding date must be after starting date!", "Error");
```

```
        else
            Message_Warning("Invalid date format!\nEnter again!", "Error");
        goToXY(50, 18);
        TextColor(63);
        cout << "                                                ";
        goToXY(52, 18);
        cout << "Ending date (dd/mm/yyyy): "; // Get the date and ensure its format
        tmp = Limit_Input(80, 18, 10, 63);
    }
    // sem_cur->endDate = tmp;
    date2 = tmp;
    sem_cur->startDate = date1;
    sem_cur->endDate = date2;
    // if (compareDate(date1, date2) == 1)
    // {
    //     Message_Warning("Invalid date!\nEnding date must be after starting date!", "Error");
    //     return AddSemester(yearHead);
    // }

    int Y = year_cur->yearStart;
    string out_year = to_string(Y) + '_' + to_string(Y + 1);
    string outPath = "..\\Data_file\\" + out_year + "\\smt" + to_string(N);
    string tmp_sys = "mkdir " + outPath;
    // Create a folder to store its information
    const char *cstr_path = tmp_sys.c_str();
    TextColor(7);
    goToXY(30, 25);
    system(cstr_path);
    goToXY(30, 25);
    cout << "                                                                ";
    return sem_cur;
}
```

- “isStringDigits” takes an inputted string and check whether is contains only number of not.

```
bool isStringDigits(string str)
{
    for (char ch : str)
        if (!std::isdigit(ch))
            return false;
    return true;
}
```

- File: [Sem]_Interface_New_Stuff.cpp has the 2 functions which call the corresponding functions.
  - "New_Stuff" shows the menu that the program allows users to choose to add to the program, which is to add a new year, a new semester, or a new course.

```
360  void New_Stuff(Year *yearHead, Account* accHead)
361  {
362      system("cls");
363      Render_NewInfo(55, 3);
364
365      vector<string> menu;
366      menu.push_back("Add a new year");
367      menu.push_back("Add a new semester");
368      menu.push_back("Add a new course");
369      menu.push_back("Back to main menu");
370
371      int ye;
372      Year *year_cur = nullptr;
373
374      int opt = Draw_XY(menu, 60, 12, 4, 24, 63);
375      switch (opt)
376      {
377      case 0:
378          //? Add a new year
379          Interface_New_Year(yearHead, accHead);
380          return;
381      case 1:
382          //? Add a new semester
383          system("cls");
384          Interface_New_Sem(yearHead);
385          // Recursion back to the StaffMain function
386          return;
387      case 2:
388          //? Add a new course
389          ye = Get_CheckFormat_Existed_Year(yearHead);
390          year_cur = yearHead;
391          while (year_cur && year_cur->yearStart != ye)
392              year_cur = year_cur->next;
393          AddingCourse(year_cur->NoSemester, yearHead);
394          // Recursion back to the StaffMain function
395          return;
396      case 3:
397          return;
398      }
399  }
```

- "Interface_New_Sem" gives the option for the user to use the follow-up function after making the addition of a new semester.

```
402  Semester *Interface_New_Sem(Year *yearHead)
403  {
404      Semester *semCurrent = AddSemester(yearHead); // New semester and return the default for next actions
405      string mess;
406      mess = "Do you want to add a course to this semester?";
407      if (Message_YesNo(mess, "Notice"))
408          AddingCourse(semCurrent, yearHead);
409
410      system("cls");
411      mess = "Do you want to add another semester?";
412      if (Message_YesNo(mess, "Notice"))
413          Interface_New_Sem(yearHead);
414      return semCurrent;
415  }
```

- File: [Sem]_Initialize_Modify.cpp has the 2 functions which call the corresponding functions.
    - "initModify" function shows the options for the user to modify the selected item, which is semesters, courses, students, and scoreboards.

```
6   void initModify(Year *yearHead)
7   {
8       system("cls");
9
10      Render_Menu(59, 3);
11      vector<string> menu;
12      menu.push_back("Modify semester");
13      menu.push_back("Modify course");
14      menu.push_back("Remove a course");
15      menu.push_back("Add/Remove student");
16      menu.push_back("Update Student's ScoreBoard");
17      menu.push_back("Back");
18
19      int opt = Draw_XY(menu, 60, 12, 6, 30, 63);
20
21      RunModify(yearHead, opt); // Begin to do the chosen option
22      return;
23  }
```

    - "RunModify" takes in the selected option and calls the corresponding function to modify the selected item.

```
25  void RunModify(Year *yearHead, int opt)
26  {
27      system("cls");
28      Render_Menu(59, 3);
29      string id, strYear;
30      int sem, year;
31      vector<string> small_menu;
32      switch (opt)
33      {
34      case 0: // Modify the semester follow year and no_sem
35          year = Get_CheckFormat_Existed_Year(yearHead);
36          small_menu.push_back("1st semester");
37          small_menu.push_back("2nd semester");
38          small_menu.push_back("3rd semester");
39          small_menu.push_back("Back");
40          system("cls");
41          Render_Semester(59, 3);
42
43          sem = Draw_XY(small_menu, 60, 12, 4, 20, 63) + 1;
44
45          modifySemester(yearHead, year, sem);
46          break;
47      case 1: // View all the course and modify a course in the school year
48          // ViewCourse(yearHead);
49          year = Get_CheckFormat_Existed_Year(yearHead);
50          modifyCourse(yearHead, year);
51          break;
52      case 2: // Remove out the course
53          year = Get_CheckFormat_Existed_Year(yearHead);
54          removeCourse(yearHead, year);
55          break;
56      case 3: // add or remove a student from a course
57          // ViewCourse(yearHead);
58          year = Get_CheckFormat_Existed_Year(yearHead);
59          addRemoveStudent(yearHead, year);
60          break;
61      case 4:
62          UpdateStudentResult(yearHead);
63          break;
64      case 5:
65          string mess = "Modification completed!\nRemember to SAVE changes.";
66          Message_Warning(mess, "Notice");
67          return;
68      }
69
70      string mess = "Do you want to continue modifying?";
71      if (Message_YesNo(mess, "Confirm"))
72      {
73          initModify(yearHead);
74          return;
75      }
76      else
77      {
78          string mess = "Modification completed!\nRemember to SAVE changes.";
79          Message_Warning(mess, "Notice");
80          return;
81      }
82  }
```

- File: [Sem]_Mod_Course.cpp: has the functions to enable modifying course information, or remove it.
  - "modifyCourse": first it finds the chosen year node, then allows the user to input the course ID, then perform the modification in "ChangeCourseInfo" function and notices to the user whether the modification is successful or the course is not found.

```
74  void modifyCourse(Year *yearHead, int year)
75  {
76      system("cls");
77      Render_ViewCourse(50, 1);
78      Year *year_cur = yearHead;
79      while (year_cur && year_cur->yearStart != year) // move to the exact year
80          year_cur = year_cur->next;
81
82      Semester *sem_cur = year_cur->NoSemester;
83      Course *cour_cur = sem_cur->course;
84
85      string course_id;
86      TextColor(7);
87      goToXY(52, 7);
88      cout << "Course ID";
89      TextColor(63);
90      for (int i = 0; i < 3; i++)
91      {
92          goToXY(50, 8 + i);
93          cout << "                    ";
94      }
95      course_id = Limit_Input(54, 9, 10, 63);
96
97      while (cour_cur)
98      {
99          if (cour_cur->CourseID == course_id)
100         {
101             ChangeCourseInfo(cour_cur); // Change the course in4
102             Message_Warning("Course has been updated", "Sucess!");
103             return;
104         }
105         cour_cur = cour_cur->next;
106     }
107
108     if (Message_YesNo("Course not found!\nRetry?", "Notice"))
109         modifyCourse(yearHead, year); // Loop again
110     return;
111 }
```

  - "ChangeCourseInfo" shows the current course's information and has the user to choose which information they want to modify and then update to the course node.

```
113  void ChangeCourseInfo(Course *cour_cur)
114  {
115      system("cls");
116      Render_ViewCourse(50, 1);
117
118      vector<string> menu;
119      menu.push_back("Course ID: " + cour_cur->CourseID);
120      menu.push_back("Course Name: " + cour_cur->Name);
121      menu.push_back("Teacher Name: " + cour_cur->TeacherName);
122      menu.push_back("Number of credits: " + to_string(cour_cur->Credits));
123      menu.push_back("Maximum of students: " + to_string(cour_cur->maxStudents));
124      menu.push_back("Room: " + cour_cur->Room);
125      menu.push_back("Teaching day: " + cour_cur->Day);
126      menu.push_back("Course section: " + cour_cur->Session);
127
128      //! Line 9
129      vector<int> choice(menu.size(), 0);
130      int cur = 0;
131      bool stop = false;
132      while (!stop)
133      {
134          choice[cur] = 1;
135          for (int i = 0; i < menu.size(); i++)
136          {
137              int tmp_color = (choice[i] == 1 ? 63 : 7);
138
139              TextColor(tmp_color);
140              for (int j = 0; j < 3; j++)
141              {
142                  goToXY(50, 9 + i * 3 + j);
143                  cout << "                                                    ";
144              }
145              goToXY(50 + 2, 9 + i * 3 + 1);
146              cout << menu[i];
147          }
148          int tmp;
149          if (tmp = _getch())
150          {
151              switch (tmp)
152              {
```

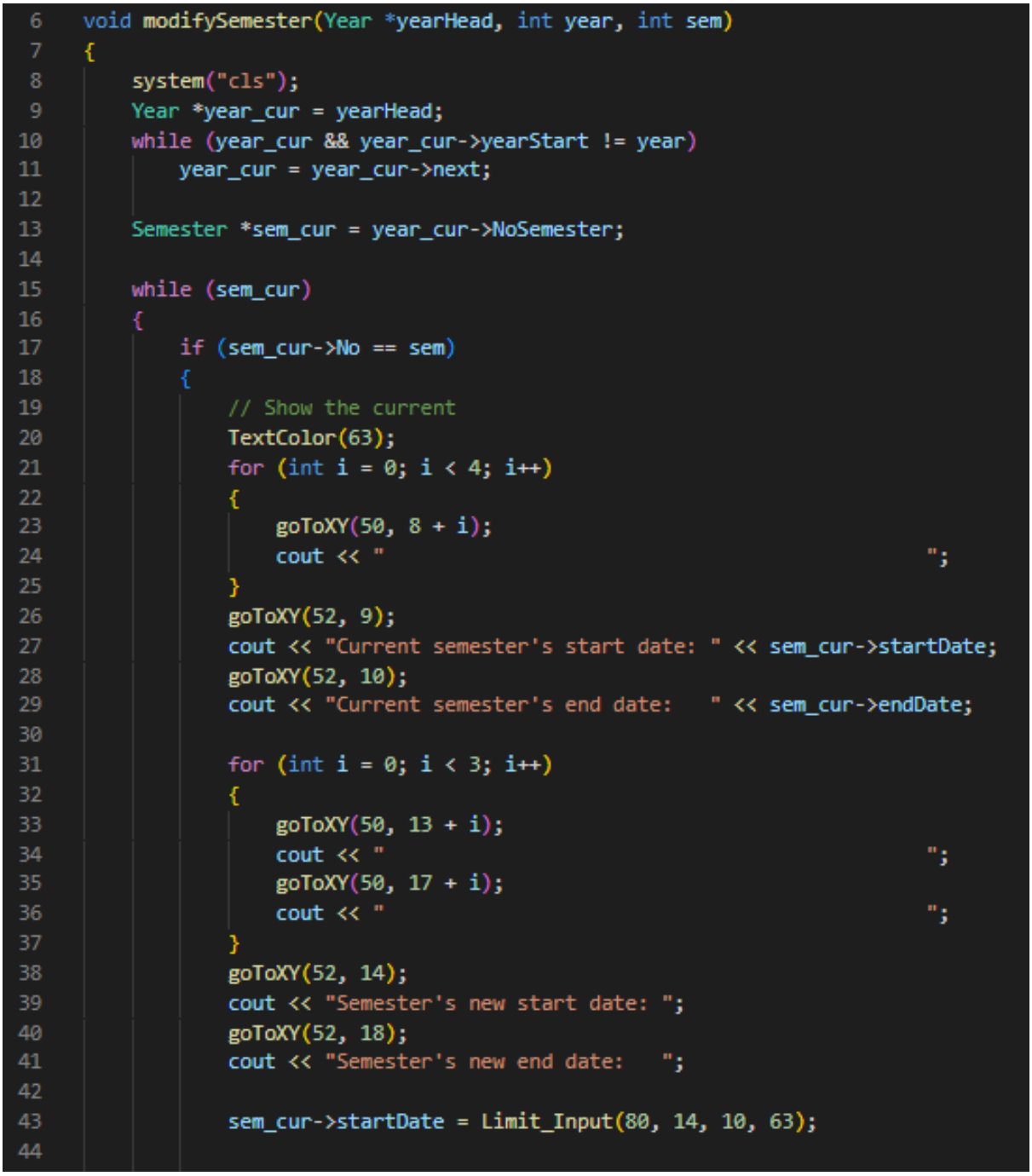
```
153             case 72:
154                 choice[cur] = 0;
155                 cur = (cur > 0 ? cur - 1 : menu.size() - 1);
156                 break;
157             case 80:
158                 choice[cur] = 0;
159                 cur = (cur < menu.size() - 1 ? cur + 1 : 0);
160                 break;
161             case 13:
162                 stop = true;
163                 break;
164             }
165         }
166     }
167     for (int i = 0; i < menu.size(); i++)
168     {
169         if (i == cur)
170         {
171             TextColor(0x9F);
172             for (int j = 0; j < 3; j++)
173             {
174                 goToXY(50, 9 + i * 3 + j);
175                 for (int k = 0; k < 50; k++)
176                     cout << " ";
177             }
178             goToXY(50 + 2, 9 + i * 3);
179             cout << menu[i].substr(0, 14);
180         }
181         else
182         {
183             TextColor(7);
184             goToXY(50 + 2, 9 + i * 3 + 1);
185             cout << menu[i];
186         }
187     }
188     string tmp;
189     switch (cur)
190     {
191     case 0:
```

```
192         cour_cur->CourseID = Limit_Input(50 + 4, 9 + cur * 3 + 1, 10, 71);
193         menu[0] = "Course ID: " + cour_cur->CourseID;
194         break;
195     case 1:
196         cour_cur->Name = Limit_Input(50 + 4, 9 + cur * 3 + 1, 35, 71);
197         menu[1] = "Course Name: " + cour_cur->Name;
198         break;
199     case 2:
200         cour_cur->TeacherName = Limit_Input(50 + 4, 9 + cur * 3 + 1, 35, 71);
201         menu[2] = "Teacher Name: " + cour_cur->TeacherName;
202         break;
203     case 3:
204         cour_cur->Credits = stoi(Limit_Input(50 + 4, 9 + cur * 3 + 1, 1, 71));
205         menu[3] = "Number of credits: " + to_string(cour_cur->Credits);
206         break;
207     case 4:
208         cour_cur->maxStudents = stoi(Limit_Input(50 + 4, 9 + cur * 3 + 1, 3, 71));
209         menu[4] = "Maximum of students: " + to_string(cour_cur->maxStudents);
210         break;
211     case 5:
212         cour_cur->Room = Limit_Input(50 + 4, 9 + cur * 3 + 1, 3, 71);
213         menu[5] = "Room: " + cour_cur->Room;
214         break;
215     case 6:
216         cour_cur->Day = Limit_Input(50 + 4, 9 + cur * 3 + 1, 3, 71);
217         while (!isDay(cour_cur->Day))
218         {
219             Message_Warning("Invalid day!\n(MON/TUE/WED/THU/FRI/SAT/SUN)", "Notice");
220             goToXY(50, 9 + cur * 3 + 1);
221             cout << "                                                    ";
222             cour_cur->Day = Limit_Input(50 + 4, 9 + cur * 3 + 1, 3, 71);
223         }
224         menu[6] = "Teaching day: " + cour_cur->Day;
225         break;
226     case 7:
227         cour_cur->Session = Limit_Input(50 + 4, 9 + cur * 3 + 1, 2, 71);
228         while (!isSession(cour_cur->Session))
229         {
230             Message_Warning("Invalid session!\n(S1/S2/S3/S4)", "Notice");
231             goToXY(50, 9 + cur * 3 + 1);
232             cout << "                                                    ";
233             cour_cur->Session = Limit_Input(50 + 4, 9 + cur * 3 + 1, 2, 71);
234         }
235         menu[7] = "Course section: " + cour_cur->Session;
236         break;
237     }
238 
239     if (Message_YesNo("Do you want to change another information?", "Notice"))
240         ChangeCourseInfo(cour_cur); // Loop again
241     return;
242 }
```

- "removeCourse" first find the chosen year node, then gets the inputting course ID and checks whether it exists or not, if it exists, remove the course node and notify the user whether the course is removed successfully or not found.

```
void removeCourse(Year *yearHead, int year)
{
    system("cls");
    Year *year_cur = yearHead;

    while (year_cur) // Find the year
    {
        if (year_cur->yearStart == year)
            break;
        year_cur = year_cur->next;
    }

    Semester *sem_cur = year_cur->NoSemester;

    if (!sem_cur) // Check if the year has any semester
    {
        string mess = "There is no semester in this year";
        Message_Warning(mess, "Notice");
        return;
    }

    Course *cour_cur = sem_cur->course, *cour_prev = nullptr;
    Render_ViewCourse(50, 1);
    // ViewCourseInYear(sem_cur);
    ViewCourse(year_cur);
    string course_id;
    TextColor(71);
    for (int i = 0; i < 3; i++)
    {
        goToXY(52, 9 + i);
        cout << "                                    ";
    }
    course_id = Limit_Input(54, 10, 10, 71);

    Semester *tmp = sem_cur;
    while (sem_cur)
    {
        while (cour_cur)
        {
            if (cour_cur->CourseID == course_id)
            {
                string mess = cour_cur->Name + " has been deleted";
                Message_Warning(mess, "Success");

                if (cour_cur == sem_cur->course)
                    sem_cur->course = cour_cur->next;
                else
                    cour_prev->next = cour_cur->next;
                delete cour_cur;

                system("cls");
                goToXY(0, 0);
                ViewCourse(year_cur);
                return;
            }
            cour_prev = cour_cur;
            cour_cur = cour_cur->next;
        }
        sem_cur = sem_cur->next;
        cour_cur = sem_cur->course;
    }

    if (Message_YesNo("Course not found, do you want to rety?", "Notice"))
        removeCourse(yearHead, year); // Loop again
    return;
}
```

- File: [Sem]_Mod_Sem_Student.cpp contains functions that make modifications to semester and enrolled students in a course.
    - "modifySemester" takes in the full year linked list and traverses through it to get the needed year and performs modification of the semester's start and end date.



```
6   void modifySemester(Year *yearHead, int year, int sem)
7   {
8       system("cls");
9       Year *year_cur = yearHead;
10      while (year_cur && year_cur->yearStart != year)
11          year_cur = year_cur->next;
12
13      Semester *sem_cur = year_cur->NoSemester;
14
15      while (sem_cur)
16      {
17          if (sem_cur->No == sem)
18          {
19              // Show the current
20              TextColor(63);
21              for (int i = 0; i < 4; i++)
22              {
23                  goToXY(50, 8 + i);
24                  cout << "                                                  ";
25              }
26              goToXY(52, 9);
27              cout << "Current semester's start date: " << sem_cur->startDate;
28              goToXY(52, 10);
29              cout << "Current semester's end date:   " << sem_cur->endDate;
30
31              for (int i = 0; i < 3; i++)
32              {
33                  goToXY(50, 13 + i);
34                  cout << "                                                  ";
35                  goToXY(50, 17 + i);
36                  cout << "                                                  ";
37              }
38              goToXY(52, 14);
39              cout << "Semester's new start date: ";
40              goToXY(52, 18);
41              cout << "Semester's new end date:   ";
42
43              sem_cur->startDate = Limit_Input(80, 14, 10, 63);
44
```

```
45              while (!isValidDate(sem_cur->startDate))
46              {
47                  if (!Message_YesNo("Invalid date, enter again: ", "Error!"))
48                      return;
49                  goToXY(50, 14);
50                  cout << "                                                  ";
51                  sem_cur->startDate = Limit_Input(80, 14, 10, 63);
52              }
53
54              sem_cur->endDate = Limit_Input(80, 18, 10, 63);
55              while (!isValidDate(sem_cur->endDate))
56              {
57                  if (!Message_YesNo("Invalid date, enter again: ", "Error!"))
58                      return;
59                  goToXY(50, 18);
60                  cout << "                                                  ";
61                  sem_cur->endDate = Limit_Input(80, 18, 10, 63);
62              }
63
64              Message_Warning("Semester has been updated", "Sucess!");
65              return;
66          }
67          sem_cur = sem_cur->next;
68      }
69
70      Message_Warning("Semester not found!", "Notice");
71      return;
72  }
```

- "addStudent" takes in the current course and the course ID to begin adding new students to the course by entering their student ID.

```
313  void addStudent(Course *courCurrent, string course_id)
314  {
315      StudentCourse *stud_cur;
316
317      if (courCurrent->numStudents == courCurrent->maxStudents)
318      {
319          Message_Warning("Course is full, cannot add more student", "Notice");
320          return;
321      }
322
323      TextColor(14);
324      goToXY(60, 17);
325      cout << "Student ID (-1 to stop)";
326      string line;
327      int i = 0;
328
329      stud_cur = courCurrent->studentCourse;
330      while (stud_cur && stud_cur->next)
331          stud_cur = stud_cur->next;
332      while (1)
333      {
334          TextColor(63);
335          goToXY(58, 19 + i);
336          cout << "                              ";
337          line = Limit_Input(60, 19 + i, 8, 63);
338          if (line == "-1")
339              break;
340          for (auto i : line)
341              if (!isdigit(i))
342              {
343                  Message_Warning("Invalid ID", "Notice");
344                  continue;
345              }
346          i++;
347          if (!stud_cur)
348          {
349              stud_cur = new StudentCourse;
350              courCurrent->studentCourse = stud_cur;
351          }
352          else
353          {
354              stud_cur->next = new StudentCourse;
355              stud_cur->next->prev = stud_cur;
356              stud_cur = stud_cur->next;
357          }
358          stud_cur->ID = line;
359          courCurrent->numStudents++;
360      }
361      Message_Warning("Finished adding student", "Success");
362      return;
363  }
```

- "removeStudent" other hand gets the course linked list and the course ID itself to remove a specific student from that chosen course.

```
366  void removeStudent(Course *courCurrent, string course_id)
367  {
368      StudentCourse *stud_cur, *stud_prev;
369      stud_cur = courCurrent->studentCourse;
370      if (!stud_cur)
371      {
372          Message_Warning("There is no student in this course", "Notice");
373          return;
374      }
375      stud_prev = stud_cur;
376
377      string ID;
378      TextColor(71);
379      for (int i = 0; i < 3; i++)
380      {
381          goToXY(52, 9 + i);
382          cout << "                                        ";
383      }
384      ID = Limit_Input(54, 10, 10, 71);
385
386      while (stud_cur)
387      {
388          if (stud_cur->ID == ID)
389          {
390              if (stud_cur == courCurrent->studentCourse)
391                  courCurrent->studentCourse = stud_cur->next;
392              else
393                  stud_prev->next = stud_cur->next;
394              delete stud_cur;
395              courCurrent->numStudents--; // Decline down the number
396              if (Message_YesNo("Student id " + ID + " has been removed", "Notice"))
397                  removeStudent(courCurrent, course_id);
398              return;
399          }
400          stud_prev = stud_cur;
401          stud_cur = stud_cur->next;
402      }
403      Message_Warning("Student not found, try again.", "Notice");
404      return;
405  }
```

- "addRemoveStudent" is the function that allows users to choose whether they want to add or remove students from the course.

```
void addRemoveStudent(Year *yearHead, int year)
{
    system("cls");
    Year *year_cur = yearHead;
    while (year_cur && year_cur->yearStart != year)
        year_cur = year_cur->next;

    Semester *sem_cur = year_cur->NoSemester;
    if (!sem_cur)
    {
        Message_Warning("No semester in this year yet!", "Notice");
        return;
    }

    vector<string> menu;
    menu.push_back("Add student to a course");
    menu.push_back("Remove student from a course");
    menu.push_back("Back to menu");

    int choice = Draw_XY(menu, 50, 10, 3, 30, 63);
    if (choice == 2)
        return;
    Message_Warning("You chose to " + string(choice == 0 ? "ADD" : "REMOVE") + " student from a course", "Notice");

    Course *cour_cur = sem_cur->course;

    string course_id;
    TextColor(63);
    for (int i = 0; i < 3; i++)
    {
        goToXY(50, 23 + i);
        cout << "                                                  ";
    }
    cout << "Input the course id that you want to add/remove student(s): ";
    course_id = Limit_Input(52, 24, 10, 63);

    while (cour_cur) // Find the course
    {
        if (cour_cur->CourseID == course_id)
        {
            system("cls");
            switch (choice) // Run the chosen option
            {
            case 1:
                addStudent(cour_cur, course_id);
                break;
            case 2:
                removeStudent(cour_cur, course_id);
                break;
            }
            return;
        }
        cour_cur = cour_cur->next;
    }

    if (Message_YesNo("Course not found!\nRetry?", "Notice"))
        addRemoveStudent(yearHead, year); // Loop again
    return;
}
```

- File: [Sem]_View_Course.cpp: holds the option to print out courses in a year.

- "Tablize" function takes in 0 for the head, 1 for the body, and 2 for the tail to draw the horizontal line of the table using ASCII characters.

```
6   // x: 0 for head, 1 for body, 2 for tail
7   void Tablize(int x)
8   {
9       int c, l, r;
10      if (x == 0)
11      {
12          c = 194;
13          l = 195;
14          r = 180;
15      }
16      else if (x == 1)
17      {
18          c = 197;
19          l = 195;
20          r = 180;
21      }
22      else
23      {
24          c = 193;
25          l = 192;
26          r = 217;
27      }
28      cout << char(l);
29      for (int i = 0; i < 12; i++)
30          cout << char(196);
31      cout << char(c);
32      for (int i = 0; i < 19; i++)
33          cout << char(196);
34      cout << char(c);
35      for (int i = 0; i < 35; i++)
36          cout << char(196);
37      cout << char(c);
38      for (int i = 0; i < 24; i++)
39          cout << char(196);
40      cout << char(r);
41  }
```

- "Part_Of_Course" prints out the list of courses inside a table with the year. Users can use the arrow keys to navigate through the table in semesters and years.

```
Course *Part_Of_Course(Course *&curCourse, Semester *&sem_cur)
{
    Render_ViewCourse(50, 1);

    int row = 10;
    char line = char(179);

    goToXY(25, row++);
    TextColor(7);
    cout << char(218);
    for (int i = 0; i < 93; i++)
        cout << char(196);
    cout << char(191);

    goToXY(25, row++);
    cout << line;
    TextColor(0xF1);
    cout << setw(44) << "Year: " << sem_cur->Year << " - " << sem_cur->Year + 1 << setw(38) << " ";
    TextColor(WHITE);
    cout << line;

    goToXY(25, row++);
    Tablize(0);

    goToXY(25, row++);
    cout << line << "  Semester  " << line
         << setw(14) << "Course ID" << setw(6) << line
         << setw(23) << "Course name" << setw(13) << line
         << setw(18) << "Teacher name" << setw(7) << line;

    goToXY(25, row++);
    Tablize(1);

    Course *cour = curCourse;
    Course *prev = nullptr;
    while (sem_cur)
    {
        while (cour)
        {
```

```
82                  goToXY(25, row++);
83                  cout << line << setw(6) << sem_cur->No << setw(7) << line;
84
85                  cout << setw(13) << cour->CourseID << setw(7) << line;
86                  int len = cour->Name.length();
87
88                  cout << setw(len + 2) << cour->Name << setw(36 - len - 2) << line;
89
90                  len = cour->TeacherName.length();
91                  cout << setw(len + 1) << cour->TeacherName << setw(25 - len - 1) << line;
92
93                  goToXY(25, row++);
94                  if (row >= 10 + 19)
95                  {
96                      Tablize(2);
97                      return cour;
98                  }
99                  // goToXY(25, row++);
100                 if (!cour->next && !sem_cur->next)
101                     Tablize(2);
102                 else
103                     Tablize(1);
104
105                 prev = cour;
106                 cour = cour->next;
107             }
108             if (!cour)
109             {
110                 if (!sem_cur->next)
111                     return prev;
112                 sem_cur = sem_cur->next;
113                 cour = sem_cur->course;
114             }
115         }
116         return prev; //!
117     }
```

- "ViewCourse" finds the number of pages that the list of courses fits in and handles the work of getting arrow keys to navigate the course for the 'Part_Of_Course' to print the course out.

```
119  void ViewCourse(Year *yearHead)
120  {
121      // string arrow = "<<-->>";
122      Semester *sem_cur = yearHead->NoSemester;
123      Course *curCourse = nullptr;
124      int cnt = 0;
125      while (sem_cur)
126      {
127          curCourse = sem_cur->course;
128          while (curCourse)
129          {
130              cnt++;
131              curCourse = curCourse->next;
132          }
133          sem_cur = sem_cur->next;
134      }
135
136      sem_cur = yearHead->NoSemester;
137      curCourse = sem_cur->course;
138      Course *cour = Part_Of_Course(curCourse, sem_cur);
139      int ind = 1;
140      cnt = cnt / 7 + (cnt % 7 == 0 ? 0 : 1);
141      goToXY(90, 32);
142      cout << "  Page " << ind << "/" << cnt << " ";
143      if (ind != cnt)
144          cout << char(31);
145      goToXY(90, 33);
146      cout << "Enter or ESC to exit";
147      Course *tmp = nullptr;
148      Semester *semtmp = nullptr;
149
150      while (1)
151      {
152          int c = _getch();
153          switch (c)
154          {
155          case UP:
156              if (!cour)
157                  break;
```

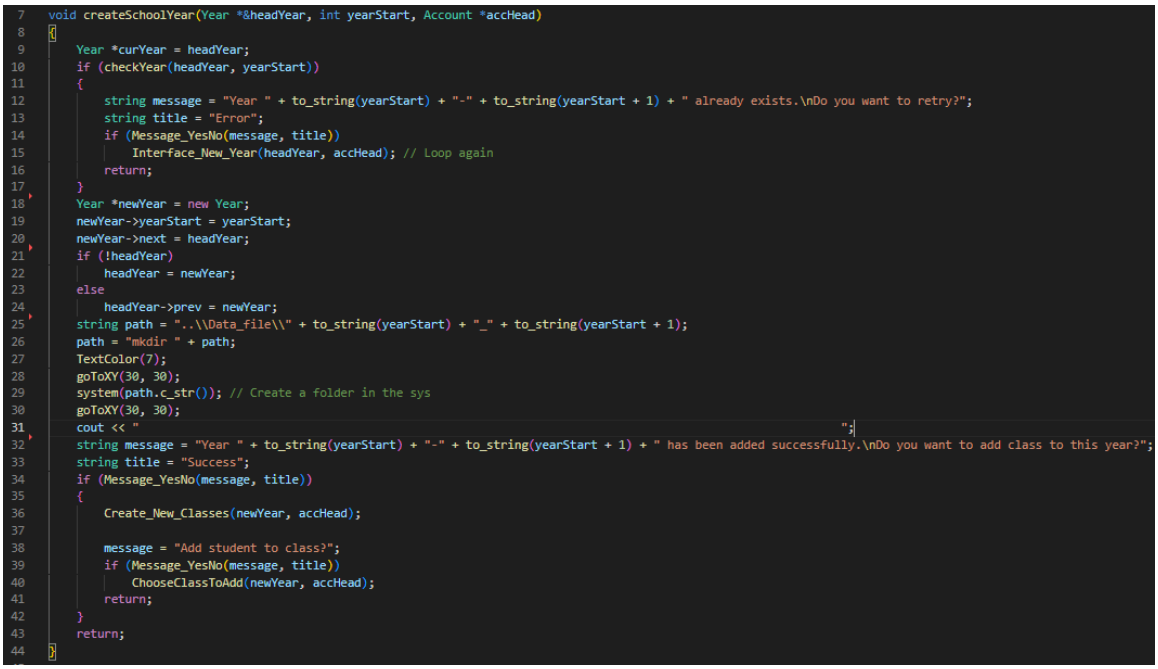
```
158             tmp = cour;
159             semtmp = sem_cur;
160             while (cour != curCourse)
161             {
162                 cour = cour->prev;
163                 if (!cour)
164                 {
165                     if (!sem_cur->prev)
166                         break;
167                     sem_cur = sem_cur->prev;
168                     cour = sem_cur->course;
169                     while (cour->next)
170                         cour = cour->next;
171                 }
172             }
173             if (!curCourse->prev && !sem_cur->prev)
174             {
175                 cour = tmp;
176                 sem_cur = semtmp;
177                 break;
178             }
179             if (curCourse->prev || sem_cur->prev)
180             {
181                 for (int i = 0; i < 7; i++)
182                 {
183 
184                     curCourse = curCourse->prev;
185                     if (!curCourse)
186                     {
187                         if (!sem_cur->prev)
188                         {
189                             cour = tmp;
190                             sem_cur = semtmp;
191                             break;
192                         }
193                         sem_cur = sem_cur->prev;
194                         curCourse = sem_cur->course;
195                         while (curCourse->next)
196                             curCourse = curCourse->next;
```

```
197                 }
198             }
199             system("cls");
200             ind--;
201             goToXY(90, 32);
202             if (ind == 1)
203                 cout << "  Page " << ind << "/" << cnt << " " << char(31);
204             else
205                 cout << char(30) << " Page " << ind << "/" << cnt << " " << char(31);
206             goToXY(90, 33);
207             cout << "Enter or ESC to exit";
208             cour = Part_Of_Course(curCourse, sem_cur);
209         }
210         break;
211     case DOWN:
212         if (!cour)
213             break;
214         if (cour->next || sem_cur->next)
215         {
216             if (!cour->next)
217             {
218                 sem_cur = sem_cur->next;
219                 curCourse = sem_cur->course;
220             }
221             else
222                 curCourse = cour->next;
223             system("cls");
224             ind++;
225             goToXY(90, 32);
226             if (ind == cnt)
227                 cout << char(30) << " Page " << ind << "/" << cnt;
228             else
229                 cout << char(30) << " Page " << ind << "/" << cnt << " " << char(31);
230             cour = Part_Of_Course(curCourse, sem_cur);
231         }
232         break;
233     case LEFT:
234         if (!yearHead->prev || !yearHead->prev->NoSemester)
235             break;
```

```
236
237             ind = 1, cnt = 0;
238             yearHead = yearHead->prev;
239             semtmp = yearHead->NoSemester;
240             while (semtmp)
241             {
242                 curCourse = semtmp->course;
243                 while (curCourse)
244                 {
245                     cnt++;
246                     curCourse = curCourse->next;
247                 }
248                 semtmp = semtmp->next;
249             }
250             cnt = cnt / 7 + (cnt % 7 == 0 ? 0 : 1);
251             system("cls");
252             goToXY(90, 32);
253             cout << "  Page " << ind << "/" << cnt << " ";
254             if (ind != cnt)
255                 cout << char(31);
256             goToXY(90, 33);
257             cout << "Enter or ESC to exit";
258
259             sem_cur = yearHead->NoSemester;
260             curCourse = sem_cur->course;
261             cour = Part_Of_Course(curCourse, sem_cur);
262             break;
263
264         case RIGHT:
265             if (!yearHead->next || !yearHead->next->NoSemester)
266                 break;
267
268             ind = 1, cnt = 0;
269             yearHead = yearHead->next;
270             semtmp = yearHead->NoSemester;
271             while (semtmp)
272             {
273                 curCourse = semtmp->course;
274                 while (curCourse)
275                 {
276                     cnt++;
277                     curCourse = curCourse->next;
278                 }
279                 semtmp = semtmp->next;
280             }
281             cnt = cnt / 7 + (cnt % 7 == 0 ? 0 : 1);
282             system("cls");
283             goToXY(90, 32);
284             cout << "  Page " << ind << "/" << cnt << " ";
285             if (ind != cnt)
286                 cout << char(31);
287             goToXY(90, 33);
288             cout << "Enter or ESC to exit";
289
290             sem_cur = yearHead->NoSemester;
291             curCourse = sem_cur->course;
292             cour = Part_Of_Course(curCourse, sem_cur);
293             break;
294
295         case ENTER:
296             return;
297         case ESC:
298             return;
299         }
300     }
301 }
```

iii. [Sem] Year

- File: [Year]_Add_New_Year_Classes.cpp: holds functions that create new year node and classes addition operations
    - "createSchoolYear" first checks the year, then if not create a new year node at the end and create a new year path in the saving folder.



```
void createSchoolYear(Year *&headYear, int yearStart, Account *accHead)
{
    Year *curYear = headYear;
    if (checkYear(headYear, yearStart))
    {
        string message = "Year " + to_string(yearStart) + "-" + to_string(yearStart + 1) + " already exists.\nDo you want to retry?";
        string title = "Error";
        if (Message_YesNo(message, title))
            Interface_New_Year(headYear, accHead); // Loop again
        return;
    }
    Year *newYear = new Year;
    newYear->yearStart = yearStart;
    newYear->next = headYear;
    if (!headYear)
        headYear = newYear;
    else
        headYear->prev = newYear;
    string path = "..\\Data_file\\" + to_string(yearStart) + "_" + to_string(yearStart + 1);
    path = "mkdir " + path;
    TextColor(7);
    goToXY(30, 30);
    system(path.c_str()); // Create a folder in the sys
    goToXY(30, 30);
    cout << "                                                                                ";
    string message = "Year " + to_string(yearStart) + "-" + to_string(yearStart + 1) + " has been added successfully.\nDo you want to add class to this year?";
    string title = "Success";
    if (Message_YesNo(message, title))
    {
        Create_New_Classes(newYear, accHead);

        message = "Add student to class?";
        if (Message_YesNo(message, title))
            ChooseClassToAdd(newYear, accHead);
        return;
    }
    return;
}
```

    - "Create_New_Classes" allows users to create new classes according to the newly created year and put them in the class linked list.

```cpp
void Create_New_Classes(Year *newYear, Account *accHead)
{
    system("cls");
    Render_Class(50, 3);
    goToXY(63, 15);
    TextColor(0x0E);
    cout << "YEAR: " << newYear->yearStart << "_" << newYear->yearStart + 1;
    goToXY(60, 17);
    cout << "New Class (-1 to stop)";
    Class *ClassTMP = newYear->Class;
    string line;
    int i = 0;
    while (1)
    {
        if (i == 10) // 10 classes per page when input
        {
            i = 0;
            system("cls");
            Render_Class(50, 3);
            goToXY(63, 15);
            TextColor(0x0E);
            cout << "YEAR: " << newYear->yearStart << "_" << newYear->yearStart + 1;
            goToXY(60, 17);
            cout << "New Class (-1 to stop)";
            TextColor(63);
            goToXY(58, 19 + i);
            cout << "                              ";
            goToXY(65, 19 + i);
            cout << line;
            i++;
        }

        TextColor(63);
        goToXY(58, 19 + i);
        cout << "                              ";
        line = Limit_Input(65, 19 + i, 8, 63);

        if (line == "-1")
            break;
        CapitalClassName(line); // Capitalize the first letter of each word
        if (checkClass(newYear, line))
        {
            string message = "Class " + line + " already exists.\nDo you want to retry?";
            string title = "Error";
            if (Message_YesNo(message, title))
                continue;
            else
                break;
        }
        if (line.substr(0, 2) != to_string(newYear->yearStart).substr(2, 2)) // Check if the class name is valid
        {
            string message = "Invalid class name.\nDo you want to retry?";
            string title = "Error";
            if (Message_YesNo(message, title))
                continue;
            else
                break;
        }
        i++;
        if (!ClassTMP)
        {
            ClassTMP = new Class;
            newYear->Class = ClassTMP;
        }
        else
        {
            ClassTMP->next = new Class;
            ClassTMP->next->prev = ClassTMP;
            ClassTMP = ClassTMP->next;
        }
        ClassTMP->Name = line;
        string message = "Added class " + line + " successfully.";
        Message_Warning(message, "Success");
    }
    return;
}
```

- "ChooseClassToAdd" prints out the classes for users to choose to begin adding students.

```
132 void ChooseClassToAdd(Year *curYear, Account *accHead)
133 {
134     system("cls");
135     TextColor(7);
136 
137     Render_Class(50, 1);
138     vector<string> listClass;
139     Class *class_cur = curYear->Class;
140     while (class_cur)
141     {
142         listClass.push_back(class_cur->Name);
143         class_cur = class_cur->next;
144     }
145     goToXY(60, 12);
146     cout << "Classes in " << curYear->yearStart << "-" << curYear->yearStart + 1;
147 
148     // int opt = Draw_ShortVer(listClass, 60, 12, 63);
149     int opt = Draw_XY(listClass, 60, 12, 4, 22, 63);
150     string className = listClass[opt];
151 
152     class_cur = curYear->Class;
153     while (class_cur)
154     {
155         if (class_cur->Name == className)
156             break;
157         class_cur = class_cur->next;
158     }
159     Method(accHead, class_cur);
160     string mess = "Do you want to add student to another class?";
161     if (Message_YesNo(mess, "Another student?"))
162     {
163         ChooseClassToAdd(curYear, accHead);
164         return;
165     }
166     return;
167 }
```

- "Method" contains the style the users want to add students either by hand or from the file.

```
170  void Method(Account *accHead, Class *curClass)
171  {
172      system("cls");
173      Render_Student(35, 3);
174      vector<string> menu;
175      menu.push_back("Adding by hand");
176      menu.push_back("Importing from file");
177      menu.push_back("Return");
178
179      TextColor(63);
180      for (int i = 0; i < 3; i++)
181      {
182          goToXY(58, 11 + i);
183          cout << "                                    ";
184      }
185      goToXY(60, 12);
186      cout << "Choose method to add student";
187
188      int opt = Draw_ShortVer(menu, 60, 15);
189      switch (opt)
190      {
191      case 0:
192          inputStudent(accHead, curClass);
193          return;
194      case 1:
195          importStudent(accHead, curClass);
196          return;
197      }
198      return;
199  }
```

- "inputStudent" begins to print out the list of information that needs to be filled in.

```
202  void inputStudent(Account *accHead, Class *curClass)
203  {
204      system("cls");
205      Render_Student(45, 3);
206      string ID, first, last, gen, birth, socialID;
207      string yr = curClass->Name.substr(0, 2);
208
209      Draw_In_Stud(ID, first, last, gen, birth, socialID, yr);
210
211      if (Check_Student(curClass, ID))
212      {
213          string message = "Student has already been added!!!\nDo you want to re-enter?";
214          if (Message_YesNo(message, "Conitnue"))
215              inputStudent(accHead, curClass);
216          return;
217      }
218      //? Add here
219      add1stYearStudents(accHead, curClass, ID, first, last, gen, birth, socialID); // add to class
220      string message = "Student " + last + " " + first + " has been added to class " + curClass->Name + ".";
221      message += "\nDo you want to add another to this class?";
222
223      string title = "Student added";
224      if (Message_YesNo(message, title))
225      {
226          inputStudent(accHead, curClass);
227          return;
228      }
229      return;
230  }
```

- "importStudent" has the users choose the file they want to import new students to classes and perform operations for the data to fit in the linked list.

```
233  void importStudent(Account *accHead, Class *curClass)
234  {
235      system("cls");
236      int y = 3;
237      Render_Student(45, y);
238      goToXY(50, 12);
239      cout << "Please import the CSV file to folder New_Enrolled_Student.\n";
240      goToXY(50, 13);
241      cout << "Please enter file name(default: studentlist.csv, 0: to choose default): ";
242
243      string fileName = "studentlist.csv";
244      string tmp;
245
246      goToXY(60, 14);
247      cout << "File name: ";
248      tmp = Limit_Input(60 + 11, 14, 25, 7);
249      if (tmp != "0")
250          fileName = tmp;
251      // Get the file name in4 and access the file
252      string path = "..\\Data_file" + separator + "New_Enrolled_Student" + separator + fileName;
253      goToXY(60, 16);
254      cout << "Using file at: " << path;
255      // check if the file exits
256      ifstream studentList(path.c_str());
257
258      if (!studentList.is_open())
259      {
260          string message = "File not found!!!\nDo you want to retry?";
261          if (Message_YesNo(message, "Conitnue"))
262              importStudent(accHead, curClass);
263          return;
264      }
265
266      // read CSV file line by line
267      string line;
268      string delimiter = ",";
269      int lineCount = 0;
270
271      getline(studentList, line);
272      while (getline(studentList, line))
```

```
    {
        // split the line by commas
        size_t position = 0;
        stringstream ss(line);
        string token;
        string fields[6];
        int i = 0;

        while (i < 6)
        {
            if (i < 5)
            {
                position = line.find(delimiter);
                token = line.substr(0, position);
                line.erase(0, position + delimiter.length());
            }
            else
                token = line;
            fields[i] = token;
            i++;
        }

        // save the student information
        string studentID = fields[0];
        string firstName = fields[1];
        string lastName = fields[2];
        string gender = fields[3];
        string dateofBirth = fields[4];
        string socialID = fields[5];

        lineCount++;
        // add the student to class
        if (Check_Student(curClass, studentID))
        {
            lineCount--;
            continue;
        }
        else
            add1stYearStudents(accHead, curClass, studentID, firstName, lastName, gender, dateofBirth, socialID);
    }
    string mess = "Imported " + to_string(lineCount) + " students to class " + curClass->Name + ".";
    Message_Warning(mess, "Success");
}
```

- "add1stYearStudents" uses the new student information and adds them to the new node of students in class linked list.

```
void add1stYearStudents(Account *accHead, Class *curClass, string ID, string first, string last, string gen, string birth, string socialID)
{
    Student *headStudent = curClass->StudentClass;
    Student *newStud = new Student;

    newStud->ID = ID;
    newStud->accStudent = new Account;

    //? Set information to student
    Account *accTmp = newStud->accStudent;
    accTmp->role = 1; //? Set role for student
    accTmp->firstName = first;
    accTmp->lastName = last;
    accTmp->Gender = gen;
    accTmp->birth = birth;
    accTmp->SocialID = socialID;

    if (!headStudent)
        curClass->StudentClass = newStud;
    else
    {
        while (headStudent->next)
            headStudent = headStudent->next;
        headStudent->next = newStud;
        newStud->prev = headStudent;
    }
    NewAccount(accHead, ID, accTmp);
    return;

    // string message = "Student " + accTmp->firstName + " " + accTmp->lastName + " has been added to class " + curClass->Name + ".";
    // string title = "Success";
    // Message_Warning(message, title);
}
```

- "Draw_In_Stud" prints out the boxes the user enters the student's information and checks for their validity.

```
352 void Draw_In_Stud(string &ID, string &first, string &last, string &gen, string &birth, string &socialID, string ye)
353 {
354     system("cls");
355     TextColor(3);
356     Render_Student(45, 1);
357 
358     vector<string> menu;
359     menu.push_back("Student ID:          ");
360     menu.push_back("Student fullname:    ");
361     menu.push_back("Student gender(M/F): ");
362     menu.push_back("Student birth:       ");
363     menu.push_back("Student socialID:    ");
364     string space = "                                                    ";
365 
366     TextColor(7);
367     int i;
368     for (i = 0; i < menu.size(); i++)
369     {
370         goToXY(62, 10 + 4 * i);
371         cout << menu[i];
372     }
373 
374     string line, full;
375     i = 0;
376     while (i < menu.size())
377     {
378         TextColor(63);
379         for (int j = 0; j < 3; j++)
380         {
381             goToXY(60, 9 + 4 * i + j);
382             cout << space;
383         }
384         goToXY(62, 10 + 4 * i);
385         cout << menu[i];
386 
387         if (i == 0) // ID
388         {
389             line = Limit_Input(62 + menu[i].length() - 1, 10 + 4 * i, 8, 63);
390             while (!isValidStudentID(line, ye))
```

```
            {
                string mess = "Invalid student ID. Please enter again.";
                Message_Warning(mess, "Warning");
                TextColor(63);
                goToXY(60, 9 + 4 * i + 1);
                cout << space;
                goToXY(62, 10 + 4 * i);
                cout << menu[i];
                line = Limit_Input(62 + menu[i].length() - 1, 10 + 4 * i, 8, 63);
            }
            ID = line;
        }
        else if (i == 1) // fullname
        {
            line = Limit_Input(62 + menu[i].length() - 1, 10 + 4 * i, 25, 63);
            full = line;
        }
        else if (i == 2) // Gender
        {
            line = Limit_Input(62 + menu[i].length() - 1, 10 + 4 * i, 1, 63);
            while (!isValidGender(line))
            {
                TextColor(63);
                string mess = "Invalid gender format (M: male, F:female).\n Please enter again.";
                Message_Warning(mess, "Warning");
                goToXY(60, 9 + 4 * i + 1);
                cout << space;
                goToXY(62, 10 + 4 * i);
                cout << menu[i];
                line = Limit_Input(62 + menu[i].length() - 1, 10 + 4 * i, 1, 63);
            }
            gen = line;
        }

        else if (i == 3) // birth
        {
            line = Limit_Input(62 + menu[i].length() - 1, 10 + 4 * i, 10, 63);
            while (!isValidDate(line))
            {
                TextColor(63);
                string mess = "Invalid date format (dd/mm/yyyy).\n Please enter again.";
                Message_Warning(mess, "Warning");
                goToXY(60, 9 + 4 * i + 1);
                cout << space;
                goToXY(62, 10 + 4 * i);
                cout << menu[i];
                line = Limit_Input(62 + menu[i].length() - 1, 10 + 4 * i, 10, 63);
            }
            birth = line;
        }
        else if (i == 4) // social ID
        {
            line = Limit_Input(62 + menu[i].length() - 1, 10 + 4 * i, 12, 63);
            socialID = line;
        }
        TextColor(7);
        for (int j = 0; j < 3; j++)
        {
            goToXY(60, 9 + 4 * i + j);
            cout << space;
        }
        goToXY(62, 10 + 4 * i);
        cout << menu[i] << line;
        i++;
    }
    SeparateName(full, first, last);
    return;
}
```

- File: [Year]_New_Year_Interface.cpp: stores functions with the purpose of getting back the file and having the user choose the year they want to create.
  - "Show_Year_List" prints out the lists of existing years.

```
7   void Show_Year_List(Year *yearHead)
8   {
9       Year *year_cur = yearHead;
10      system("cls");
11      TextColor(0xF1);
12      int i = 0;
13
14      goToXY(67, 11);
15      cout << "=====YEAR LIST=====";
16
17      while (year_cur)
18      {
19          int year_tmp = year_cur->yearStart;
20          goToXY(67, 12 + i++);
21          cout << "    > " << year_tmp << '-' << year_tmp + 1 << "      ";
22          year_cur = year_cur->next;
23      }
24      TextColor(WHITE);
25  }
```

  - "Get_CheckFormat_Existed_Year" first calls 'Show_Year_List" function and allows the user to input the school year.

```
28  int Get_CheckFormat_Existed_Year(Year *yearHead)
29  {
30      string strYear;
31      Show_Year_List(yearHead);
32
33      goToXY(60, 18);
34      TextColor(0x0E);
35      cout << "Choose year";
36      TextColor(63);
37      for (int i = 0; i < 3; i++)
38      {
39          goToXY(59, 19 + i);
40          cout << "                                        ";
41      }
42
43      strYear = Limit_Input(61, 20, 9, 63);
44      strYear[4] = '_';
45
46      while (!isValidYear(strYear) || !checkYear(yearHead, stoi(strYear.substr(0, 4))))
47      {
48          string message = "Invalid year format or system hadn't had this year yet!\nPlease retry.";
49          string title = "Error";
50          Message_Warning(message, title);
51          goToXY(59, 20);
52          cout << "                                        ";
53          strYear = Limit_Input(61, 20, 9, 63);
54          strYear[4] = '_';
55      }
56      TextColor(7);
57      return stoi(strYear.substr(0, 4));
58  }
```

- "RecoverFile" is called to begin loading all files that saved in files and create the first linked list of years.

```
60  Year *RecoverFile()
61  {
62      Year *headYear = nullptr;
63      int numofYear = 0;
64      // import existing years and classes
65      loadingFile(headYear, numofYear);
66      Year *yearTMP = headYear = SortYear_Descending(headYear);
67      while (yearTMP)
68      {
69          yearTMP->NoSemester = Read_All_Semester(yearTMP->yearStart); // read the semesters in years
70          readAllFileCourses(yearTMP->NoSemester);
71          yearTMP = yearTMP->next;
72      }
73      return headYear;
74  }
```

- "Interface_New_Year" allows users to enter the year they want to create and call 'create_new_year' function to create the new year.

```
76  void Interface_New_Year(Year *yearHead, Account *accHead)
77  {
78      Year *year_cur = yearHead;
79      string strYear;
80      system("cls");
81      Render_NewYear(50, 1);
82      Show_Year_List(yearHead);
83      goToXY(52, 18);
84      TextColor(0x0E);
85      cout << "Choose year (Enter -1 to back)";
86      TextColor(63);
87      for (int i = 0; i < 3; i++)
88      {
89          goToXY(50, 19 + i);
90          cout << "                                                  ";
91      }
92      strYear = Limit_Input(52, 20, 9, 63);
93      TextColor(7);
94      if (strYear == "-1")
95          return;
96      strYear[4] = '_';
97      while (!isValidYear(strYear))
98      {
99          string message = "Invalid year format or already has!\nPlease retry.";
100         string title = "Error";
101         Message_Warning(message, title);
102         TextColor(63);
103         goToXY(50, 20);
104         cout << "                                                  ";
105         strYear = Limit_Input(52, 20, 9, 63);
106         strYear[4] = '_';
107     }
108     TextColor(7);
109     int start = stoi(strYear.substr(0, 4));
110     createSchoolYear(yearHead, start, accHead);
111     return;
112 }
```

iv. [Help] Section

- File: [Help]_CaculateAmount.cpp: This file contains functions to calculate the amount of number of elements in the linked list or calculate the GPA of.
  - The “CaculateGPA_1_Student” function calculates the total score of all students, then divides it by the number of subjects studied by the student and returns the result.

```
double CaculateGPA_1_Student(double *&SBC, int n) // n là số môn, tức là số cột của bảng SBC. n là số đã trừ cột cuối ra
{
    double GPA = 0;
    int count = 0; // biến này để đếm xem học sinh này học bao nhiêu môn
    for (int i = 0; i < n; i++)
    {
        if (SBC[i] >= 0)
        {
            GPA += SBC[i];
            count++;
        }
    }
    if (count != 0)
        GPA /= count;
    return GPA;
}
```

  - The “amountClass” function to calculate the number of classes in the listed Class.

```
int amountClass(Class *classHead)
{
    int n = 0;
    while (classHead)
    {
        n++;
        classHead = classHead->next;
    }
    return n;
}
```

  - The “amountCourseOfStudent” function to calculate the number of courses of a student or an all-student in class.

```
int amountCourseOfStudent(CourseStudent *courseHead)
{
    int n = 0;
    while (courseHead)
    {
        courseHead = courseHead->next;
        n++;
    }
    return n;
}
```

- The “amountStudentOfCourse” function calculates the number of course students.

```
39  int amountStudentOfCourse(StudentCourse *studentHead)
40  {
41      int n = 0;
42      while (studentHead)
43      {
44          n++;
45          studentHead = studentHead->next;
46      }
47      return n;
48  }
```

- The “amountStudentOfClass” function calculates the number of class students.

```
50  int amountStudentOfClass(Student *studentHead)
51  {
52      int count = 0;
53      while (studentHead)
54      {
55          studentHead = studentHead->next;
56          count++;
57      }
58      return count;
59  }
```

- File: [Help]_Check.cpp: This file contains the function to check the existence of a file or an element in a linked list and to check whether the value is valid or not.
  - The “checkFile” function checks if the file exists or not, if yes, return true, return false.

```
3   bool checkFile(string name)
4   {
5       fstream f;
6       f.open(name);
7       if (f.is_open())
8           return true;
9       return false;
10  }
```

  - The “isDouble” function checks if the number is of type double.

```cpp
bool isDouble(string str)
{
    istringstream ss(str);
    double d;
    return (ss >> d) && (ss.eof());
}
```

- The "checkExistence_OfCourse" function checks if the course in the list course of students exists or not, if yes, return Cours, return nullptr.

```cpp
CourseStudent *checkExistence_OfCourse(CourseStudent *courseCheck, CourseStudent *courseHead)
{
    while (courseHead)
    {
        if (courseCheck->course == courseHead->course)
            return courseCheck;
        courseHead = courseHead->next;
    }
    return nullptr;
}
```

- File: [Help]_Choose.cpp: This file contains a function that displays choices and the user selects by pressing enter and returns the selected choices. The x and y variables are the coordinates to start showing this selection in all functions below.
    - The "chooseYearbyOption_XY": shows the years and the end of the selection list is a BACK selection. If the user selects a year, that year is returned, and if Back is selected, nullptr is returned.

```cpp
Year *chooseYearbyOption_XY(Year *yearHead, int x, int y, int nOption_eachTime)
{
    Year *year_cur = yearHead;
    vector<string> menu;
    while (year_cur)
    {
        menu.push_back(to_string(year_cur->yearStart) + " - " + to_string(year_cur->yearStart + 1));
        year_cur = year_cur->next;
    }
    int width = 20;
    menu.push_back("BACK");
    int option = Draw_XY(menu, x, y, nOption_eachTime, width, 63);

    while (yearHead && option--)
        yearHead = yearHead->next;
    return yearHead;
}
```

    - The "chooseClassrbyOption_XY": function shows the class in the year and the end of the selection list is a back selection. If the user selects a class, that class is returned, and if Back is selected, nullptr is returned.

```
Class *chooseClassbyOption_XY(Class *classHead, int x, int y, int nOption_eachTime)
{
    vector<string> menu;
    Class *class_cur = classHead;
    while (class_cur)
    {
        menu.push_back("    " + class_cur->Name);
        class_cur = class_cur->next;
    }
    int width = 20;
    menu.push_back("   BACK");
    int option = Draw_XY(menu, x, y, nOption_eachTime, width, 63);

    while (classHead && option--)
        classHead = classHead->next;
    return classHead;
}
```

- The “chooseClass_inAllYear_byOption_XY” function same as the “chooseClassrbyOption_XY” function but it shows all classes in the information system.

```
Class *chooseClass_inAllYear_byOption_XY(Year *yearHead, int x, int y, int nOption_eachTime)
{
    vector<string> menu;
    Class *class_cur = nullptr;
    Year *year_cur = yearHead;
    while (year_cur)
    {
        class_cur = year_cur->Class;
        while (class_cur)
        {
            menu.push_back("    " + class_cur->Name);
            class_cur = class_cur->next;
        }
        year_cur = year_cur->next;
    }
    int width = 20;
    menu.push_back("   BACK");
    if (menu.size() == 0)
        return nullptr;
    int option = Draw_XY(menu, x, y, nOption_eachTime, width, 63);
    year_cur = yearHead;
    while (year_cur)
    {
        class_cur = year_cur->Class;
        while (class_cur)
        {
            if (option == 0)
                return class_cur;
            class_cur = class_cur->next;
            option--;
        }
        year_cur = year_cur->next;
    }
    return class_cur;
}
```

- The “chooseSemesterbyOption_XY”: function shows the Semesters in the year and the end of the selection list is a Back selection. If the user selects any Semester, that Semester is returned, and if Back is selected, nullptr is returned.

```
73  Semester *chooseSemesterbyOption_XY(Semester *semHead, int x, int y, int nOption_eachTime)
74  {
75      vector<string> menu;
76      Semester *sem_cur = semHead;
77      while (sem_cur)
78      {
79          menu.push_back("Semester " + to_string(sem_cur->No));
80          sem_cur = sem_cur->next;
81      }
82      int width = 15;
83      menu.push_back("BACK");
84      int option = Draw_XY(menu, x, y, nOption_eachTime, width, 63);
85
86      while (semHead && option--)
87      {
88          semHead = semHead->next;
89      }
90      return semHead;
91  }
```

- The “chooseCoursebyOption_XY”: function shows the Courses and the end of the selection list is a Back selection. If the user selects any Course, that Course is returned, and if Back is selected, nullptr is returned.

```
92  Course *chooseCoursebyOption_XY(Course *courseHead, int x, int y, int nOption_eachTime)
93  {
94      vector<string> menu;
95      Course *course_cur = courseHead;
96      while (course_cur)
97      {
98          menu.push_back(course_cur->Name);
99          course_cur = course_cur->next;
100     }
101     int width = 35;
102     menu.push_back("BACK");
103     int option = Draw_XY(menu, x, y, nOption_eachTime, width, 63);
104
105     while (courseHead && option--)
106         courseHead = courseHead->next;
107     return courseHead;
108 }
```

- The “ChooseStudentOfClass” function displays a table containing three columns: No, ID, and Name. if the user selects a cell then the student of that row will be selected and if the user press ESC then return.

```
Student *chooseStudentOfClass(Class *ChooseClass, int x, int y)
{
    Student *student_cur = ChooseClass->StudentClass;
    goToXY(x, y++);
    cout << "List of students in " << ChooseClass->Name;
    int width[7];
    width[0] = 10; // chiều rộng cột NO
    width[1] = 15; // chiều rộng cột ID
    width[2] = 30; // chiều rộng Student Name;

    string *title = new string[3];
    title[0] = "   No";
    title[1] = " ID";
    title[2] = "   Student Name";

    int num_row = amountStudentOfClass(ChooseClass->StudentClass);
    int num_col = 3;

    string **table = new string *[num_row];
    for (int i = 0; i < num_row; i++)
        table[i] = new string[num_col];

    int height = 1, Row_eachTime = 7, Col_eachTime = 8;
    bool edit_Col[3] = {true, true, true};

    for (int i = 0; i < num_row; i++)
    {
        int j = 0;
        table[i][j++] = "   " + to_string(i + 1);
        table[i][j++] = " " + student_cur->ID;
        table[i][j++] = "   " + student_cur->accStudent->lastName + " " + student_cur->accStudent->firstName;
        student_cur = student_cur->next;
    }
    int x_cur = 0, y_cur = 0;

    Draw_table(table, title, num_row, num_col, width, height, x, y, Row_eachTime, Col_eachTime, edit_Col, x_cur, y_cur);

    Student *ChooseStudent = nullptr;
    if (y_cur == -1)
        return ChooseStudent;
    ChooseStudent = ChooseClass->StudentClass;
    while (y_cur--)
        ChooseStudent = ChooseStudent->next;
    return ChooseStudent;
}
```

- File:[Help]_Create.cpp: This file contains the function to create a name, a linked list, or a table as a two-dimensional array. Moreover, this file also includes a function to get the first character in a string or format a Mark.
    - The "FormatMark" function converts scores from numbers to strings. if the score is valid, then switch, if not, then change to X.

```
string FormatMark(double Mark)
{
    string Format = "";
    if (Mark == 10)
        Format = to_string(Mark).substr(0, 5);
    else if (Mark < 0 || Mark > 10)
        Format = " X";
    else
        Format = " " + to_string(Mark).substr(0, 4);
    return Format;
}
```

- The "getFirstChar" function takes the first character of each word in a string and concatenates it into a string and returns that string.

```
14  string getFirstChar(string name)
15  {
16      // char* c_name = new char[name.length()];
17      // strcpy(c_name, name.c_str());
18      char *c_name = (char *)name.c_str();
19      int n = name.length();
20      if (n == 0)
21          return name;
22      string tmp = "";
23      char space = ' ';
24      tmp += toupper(c_name[0]);
25      for (int i = 0; i < n; i++)
26          if ((int)c_name[i] == (int)space)
27          {
28              if (isdigit(c_name[i]))
29                  tmp += c_name[i];
30              else
31                  tmp += toupper(c_name[i + 1]);
32          }
33      return tmp;
34  }
```

- The “createNameFile” function "generates a path to a given file by directory name, subdirectory, filename, and file type.

```
36  string createNameFile(int year, int no_smt)
37  {
38
39      string s_year = to_string(year) + "_" + to_string(year + 1) + "\\";
40      string s_smt = "smt" + to_string(no_smt) + "\\";
41      string path = s_year + s_smt;
42      return path;
43  }
```

- The “CreateSB_ofStudent_inClass” function imports the final marks of a student's course into the transcript.

```
77  void CreateSB_ofStudent_inClass(double *&SBC, CourseStudent *courseHead, Student *studentHead)
78  {
79      StudentCourse *tmp;
80      CourseStudent *courseCheck;
81      int i = 0;
82      while (courseHead)
83      {
84          courseCheck = checkExistence_OfCourse(courseHead, studentHead->course);
85          if (courseCheck)
86          {
87              tmp = find_StudentOfCourse(studentHead->ID, courseHead->course->studentCourse);
88              SBC[i++] = tmp->ScoreBoardCourse.finalMark;
89          }
90          else
91              SBC[i++] = -1; // -1 là hs này không học môn đó
92          courseHead = courseHead->next;
93      }
94  }
```

- The "createSBC_ofClass" function imports the final marks of all students' courses into the transcript and calculates the GPA of all students.

```
95  void createSBC_ofClass(double **&SBC, CourseStudent *courseHead, Student *studentHead, int Col, int Row)
96  {
97      SBC = new double *[Row];
98      for (int i = 0; i < Row; i++)
99      {
100         SBC[i] = new double[Col + 1]; // +1 là + 1 cột cuối để lưu GPA của học sinh đó trong kì này
101         CreateSB_ofStudent_inClass(SBC[i], courseHead, studentHead);
102         SBC[i][Col] = CaculateGPA_1_Student(SBC[i], Col);
103         studentHead = studentHead->next;
104     }
105 }
```

- File: [Help]_Find.cpp: This file contains functions to search for an element on request, if not found returns nullptr.
  - The "findStudentbyID" function will find the student in the system, if found then return the student, and if not found then return nullptr.

```
Student *findStudentbyID(string IDStudent, Year *Yhead)
{
    Class *curClass;
    Student *curStudent;
    while (Yhead)
    {
        curClass = Yhead->Class;
        while (curClass)
        {
            curStudent = curClass->StudentClass;
            while (curStudent)
            {
                if (curStudent->ID == IDStudent)
                    return curStudent;
                curStudent = curStudent->next;
            }
            curClass = curClass->next;
        }
        Yhead = Yhead->next;
    }
    return nullptr;
}
```

- The “find_StudentOfCOurse” function will find the Course’s Student with Student’s ID, if yes, return that student, otherwise return nullptr.

```
StudentCourse *find_StudentOfCourse(string ID, StudentCourse *stuCourseHead)
{
    while (stuCourseHead)
    {
        if (stuCourseHead->ID == ID)
            return stuCourseHead;
        stuCourseHead = stuCourseHead->next;
    }
    return nullptr;
}
```

v. [Course] Section

- File: [Course]_Interface.cpp This file includes functions that display choices so the user can choose the functionality they want, such as viewing the list course in class.
    - The “Interface_ViewScoreBoardCourse” function will let the user choose the year, then select the semester in that year, and

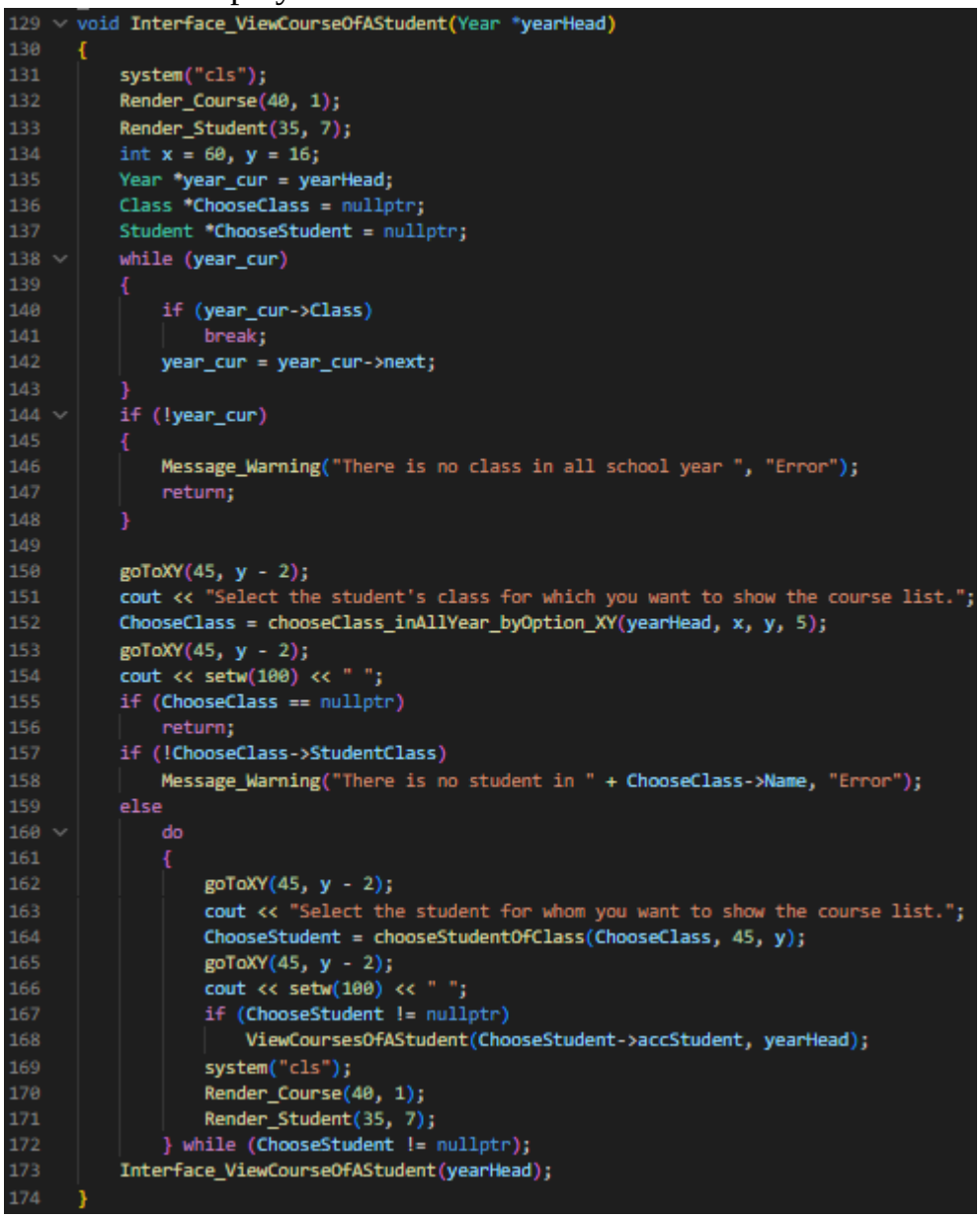
finally select the subject in that semester to display the scoreboard.

```
void Interface_ViewScoreBoardCourse(Year *yearHead) // này của task 21
{
    system("cls");

    Year *ChooseYear = nullptr;
    Semester *ChooseSem = nullptr;
    Course *ChooseCourse = nullptr;
    Render_ScoreBoardCourse();
    int x = 60, y = 17;
    if (yearHead == nullptr)
    {
        Message_Warning("There are no school year.", "Error not exist");
        // gọi lại hàm mainmenu
        return;
    }
    do
    {
        ChooseYear = chooseYearbyOption_XY(yearHead, x, y, 5);
        if (ChooseYear == nullptr)
            // quay lại main menu
            return;
        if (ChooseYear->NoSemester == nullptr)
        {
            Message_Warning("There are no semester in this school year.", "Error not exist");
            Interface_ViewCourseOfAStudent(yearHead);
            return;
        }
        ChooseSem = chooseSemesterbyOption_XY(ChooseYear->NoSemester, x, y, 5);
        if (ChooseSem == nullptr)
        {
            Interface_ViewStudentCourse(yearHead);
            return;
        }
        do
        {
            if (ChooseSem->course == nullptr)
            {
                Message_Warning("There are no course in this semester", "Error not Exist");
                break;
            }

            ChooseCourse = chooseCoursebyOption_XY(ChooseSem->course, x, y, 5);
            if (ChooseCourse == nullptr)
                break;
            if (ChooseCourse->studentCourse == nullptr)
                Message_Warning(" There is no student in course " + ChooseCourse->Name, "Error not exist");
            else
            {
                ViewScoreBoardCourse(ChooseCourse, 15, 15); // 15 15 là tọa độ x y của vị trí bắt đầu bảng
                system("cls");
                Render_ScoreBoardCourse();
            }
        } while (ChooseCourse != nullptr);
    } while (ChooseSem != nullptr);
}
```

- The “Interface_ViewStudentCourse” function allows users to choose like the function above, but after selecting, it will display a list of students in the course.

```
void Interface_ViewStudentCourse(Year *yearHead) // task 18: view students in a course
{
    system("cls");
    // in ra chư studentcourse
    Year *ChooseYear = nullptr;
    Semester *ChooseSem = nullptr;
    Course *ChooseCourse = nullptr;
    Render_StudentCourse();
    int x = 60, y = 17;
    if (yearHead == nullptr)
    {
        Message_Warning("There are no school year.", "Error not exist");
        // gọi lại hàm mainmenu
        return;
    }

    goToXY(30, y - 2);
    cout << "Select the year that contains the course, in which you want to show list student";
    ChooseYear = chooseYearbyOption_XY(yearHead, x, y, 5);
    goToXY(30, y - 2);
    cout << setw(100) << " ";
    if (ChooseYear == nullptr) // quay lại main menu
        return;
    if (ChooseYear->NoSemester == nullptr)
    {
        Message_Warning("There are no semester in this school year.", "Error not exist");
        Interface_ViewStudentCourse(yearHead);
        return;
    }
    do
    {
        goToXY(30, y - 2);
        cout << "Select the semester that contains the course, in which you want to show list student";
        ChooseSem = chooseSemesterbyOption_XY(ChooseYear->NoSemester, x, y, 5);
        goToXY(30, y - 2);
        cout << setw(100) << " ";
        if (ChooseSem == nullptr)
        {
            Interface_ViewStudentCourse(yearHead);
            return;
        }
        do
        {
            if (ChooseSem->course == nullptr)
            {
                Message_Warning("There are no course in this semester", "Error not Exist");
                break;
            }
            goToXY(30, y - 2);
            cout << "Select the course for which you want to show list student";
            ChooseCourse = chooseCoursebyOption_XY(ChooseSem->course, x, y, 5);
            goToXY(30, y - 2);
            cout << setw(100) << " ";
            if (ChooseCourse != nullptr)
            {
                if (ChooseCourse->studentCourse == nullptr)
                    Message_Warning(" There is no student in course " + ChooseCourse->Name, "Error not exist");
                else
                {
                    goToXY(40, y - 2);
                    cout << " The list of student of " << ChooseCourse->Name << " course:";
                    ViewStudentCourse(ChooseCourse, 40, y);
                }
                system("cls");
                Render_StudentCourse();
            }
        } while (ChooseCourse != nullptr);
    } while (ChooseSem != nullptr);
}
```

- The “Interface_ViewCourseOfAStudent” function will let the user choose a class in the list of classes, then select a student in that class to display the list course of that student.



```
void Interface_ViewCourseOfAStudent(Year *yearHead)
{
    system("cls");
    Render_Course(40, 1);
    Render_Student(35, 7);
    int x = 60, y = 16;
    Year *year_cur = yearHead;
    Class *ChooseClass = nullptr;
    Student *ChooseStudent = nullptr;
    while (year_cur)
    {
        if (year_cur->Class)
            break;
        year_cur = year_cur->next;
    }
    if (!year_cur)
    {
        Message_Warning("There is no class in all school year ", "Error");
        return;
    }

    goToXY(45, y - 2);
    cout << "Select the student's class for which you want to show the course list.";
    ChooseClass = chooseClass_inAllYear_byOption_XY(yearHead, x, y, 5);
    goToXY(45, y - 2);
    cout << setw(100) << " ";
    if (ChooseClass == nullptr)
        return;
    if (!ChooseClass->StudentClass)
        Message_Warning("There is no student in " + ChooseClass->Name, "Error");
    else
        do
        {
            goToXY(45, y - 2);
            cout << "Select the student for whom you want to show the course list.";
            ChooseStudent = chooseStudentOfClass(ChooseClass, 45, y);
            goToXY(45, y - 2);
            cout << setw(100) << " ";
            if (ChooseStudent != nullptr)
                ViewCoursesOfAStudent(ChooseStudent->accStudent, yearHead);
            system("cls");
            Render_Course(40, 1);
            Render_Student(35, 7);
        } while (ChooseStudent != nullptr);
    Interface_ViewCourseOfAStudent(yearHead);
}
```

- File: [Course]_Menu: This file includes functions that display menus so the user can choose the functionality they want, such as updating or viewing the list course in class.
    - The “Menu_Staff_View” function display choices that staff can view and select, including a list scoreboard, a list student, a list class, and a list course.

```
void Menu_Staff_View(Year *yearHead)
{
    system("cls");
    Render_View(50, 3);

    vector<string> menu;
    menu.push_back("ScoreBoard");
    menu.push_back("List Student");
    menu.push_back("List Class");
    menu.push_back("List Course");
    menu.push_back("Back to Main Menu");

    int opt = Draw_XY(menu, 60, 12, 5, 25, 63);
    switch (opt)
    {
    case 0:
        Menu_ScoreBoard_View(yearHead);
        break;
    case 1:
        Menu_ListStudent_View(yearHead);
        break;
    case 2:
        ViewClass(yearHead);
        break;
    case 3:
        Menu_ListCourse_View(yearHead);
        break;
    case 4:
        return;
    }
    Menu_Staff_View(yearHead);
}
```

- The “Menu_ScoreBoard_View” function allows the user to choose one of three choices, including viewing the scoreboard of all class students, viewing the scoreboard of all course students, or back to Menu Staff View.

```
void Menu_ScoreBoard_View(Year *yearHead)
{
    system("cls");
    Render_ScoreBoard(20, 3);

    vector<string> menu;
    menu.push_back("ScoreBoard Of Class");
    menu.push_back("ScoreBoard Of Course");
    menu.push_back("Back to View Menu");

    int opt = Draw_XY(menu, 60, 12, 4, 24, 63);
    system("cls");
    switch (opt)
    {
    case 0:
        Interface_ViewScoreBoardClass(yearHead);
        break;
    case 1:
        Interface_ViewScoreBoardCourse(yearHead);
        break;
    case 2:
        return;
    }
    Menu_ScoreBoard_View(yearHead);
}
```

- The “Menu_ListStudent_View” function allows the user to choose one of three choices, including viewing all class students, viewing all course students, or back to Menu Staff View.

```
void Menu_ListStudent_View(Year *yearHead)
{
    system("cls");
    Render_Student(30, 3);

    vector<string> menu;
    menu.push_back("All Student Of Class");
    menu.push_back("All Student Of Course");
    menu.push_back("Back to View Menu");

    int opt = Draw_XY(menu, 60, 12, 4, 25, 63);
    system("cls");
    switch (opt)
    {
    case 0:
        Interface_ViewStudentClass(yearHead);
        break;
    case 1:
        Interface_ViewStudentCourse(yearHead);
        break;
        ;
    case 2:
        return;
    }
    Menu_ListStudent_View(yearHead);
}
```

- The “Menu_Listcourse_View” function also has three options, viewing courses of each year's school, viewing all courses in a semester, or back to Menu Staff View.

```
86  void Menu_ListCourse_View(Year *yearHead)
87  {
88      system("cls");
89      Render_Course(40, 3);
90
91      vector<string> menu;
92      menu.push_back("All Course");
93      menu.push_back("Student's Course");
94      menu.push_back("Back to View Menu");
95
96      int opt = Draw_XY(menu, 60, 12, 4, 20, 63);
97      system("cls");
98      switch (opt)
99      {
100     case 0:
101         //?
102         ViewCourse(yearHead);
103         break;
104     case 1:
105         //?
106         Interface_ViewCourseOfAStudent(yearHead);
107         break;
108     case 2:
109         return;
110     }
111     Menu_ListCourse_View(yearHead);
112 }
```

- The “Menu_Import_Export” function also has three options, import the scoreboard of a course, export the list of students in a course, or back to Menu Staff View.

```
113 void Menu_Import_Export(Year *yearHead)
114 {
115     system("cls");
116     Render_Import(40, 2);
117     Render_Export(40, 8);
118     vector<string> menu;
119     menu.push_back("Import The ScoreBoard Of A Course");
120     menu.push_back("Export The List Of Students In A Course");
121     menu.push_back("Back to View Menu");
122 
123     int opt = Draw_XY(menu, 55, 15, 4, 42,63);
124     system("cls");
125     switch (opt)
126     {
127     case 0:
128         importScoreBoardCourse(yearHead);
129         break;
130     case 1:
131         exportListStudentCourse(yearHead);
132         break;
133     case 2:
134         return;
135     }
136     Menu_Import_Export(yearHead);
137 }
```

- File [Course]_Update.cpp: This file contains functions to modify the scoreboard of a student if it has any mistakes or enter the score into the scoreboard.
    - The “updateSBC” function will display the scoreboard of a course, then it allows the user to choose the cell which wants to update by pressing enter and entering a new score. Also, if the user press ESC, then finish the function.

```
3   void updateSBC(Course *ChooseCourse, int x, int y, int x_cur, int y_cur)
4   {
5       int width[7];
6       width[0] = 5;
7       width[1] = 12;
8       width[2] = 30;
9       width[3] = 15;
10      width[4] = 15;
11      width[5] = 15;
12      width[6] = 15;
13      int *pos = new int[7]; // vị trí bắt đầu của các cột, bắt đầu từ cột đầu là 0
14      pos[0] = 0;
15      for (int i = 1; i < 7; i++)
16      {
17          pos[i] = pos[i - 1] + width[i - 1];
18      }
19      x = 10;
20      y = 15;
21      StudentCourse *studentHead = ChooseCourse->studentCourse;
22
23      int num_row = 0;
24      while (studentHead)
25      {
26          num_row++;
27          studentHead = studentHead->next;
28      }
29      if (num_row == 0)
30      {
31          Message_Warning("There are no student in course", "Notice");
32          return;
33      }
34      string **table = new string *[num_row];
35      for (int i = 0; i <= num_row; i++)
36          table[i] = new string[7];
37
38      int height = 1, Row_eachTime = 7, Col_eachTime = 7;
39      bool edit_Col[7] = {false, false, false, true, true, true, true};
40      int t = 0;
41      string *title = new string[7];
```

```
    string *title = new string[7];
    title[t++] = "NO";
    title[t++] = "ID";
    title[t++] = "Full Name";
    title[t++] = "Mid Mark";
    title[t++] = "Final Mark";
    title[t++] = "Other Mark";
    title[t++] = "Total Mark";

    studentHead = ChooseCourse->studentCourse;
    for (int i = 0; studentHead != nullptr; i++)
    {
        int j = 0;
        table[i][j++] = to_string(i + 1);
        table[i][j++] = studentHead->ID;
        table[i][j++] = studentHead->FullName;
        table[i][j++] = FormatMark(studentHead->ScoreBoardCourse.midMark);
        table[i][j++] = FormatMark(studentHead->ScoreBoardCourse.finalMark);
        table[i][j++] = FormatMark(studentHead->ScoreBoardCourse.otherMark);
        table[i][j++] = FormatMark(studentHead->ScoreBoardCourse.totalMark);
        studentHead = studentHead->next;
    }
    goToXY(x, y - 2);
    cout << "Press Enter to select";
    Draw_table(table, title, num_row, 7, width, height, x, y, Row_eachTime, Col_eachTime, edit_Col, x_cur, y_cur);
    if (x_cur == -1)
    {
        for (int i = 0; i < num_row; i++)
            delete[] table[i];
        delete[] table;
        return;
    }
    studentHead = ChooseCourse->studentCourse;
    for (int i = 0; i < y_cur; i++)
        studentHead = studentHead->next;
    int mark = 1;
    bool stop = false;
    string s_mark;
    do
    {
        goToXY(x + pos[x_cur % Col_eachTime] + 1, y + (y_cur % Row_eachTime) * (height + 1) + 2);
        TextColor(0xF3);
        cout << setw(13) << " ";
        s_mark = Limit_Input(x + pos[x_cur % Col_eachTime] + 3, y + (y_cur % Row_eachTime) * (height + 1) + 2, 11, 0xF3); // +2 để bỏ qua title
        if (isDouble(s_mark))
        {
            mark = stod(s_mark);
            if (mark < 0 || mark > 10)
                stop = !Message_YesNo("Please re-enter your score. 0 <= mark <= 10\n Press yes to re-enter or no to stop.", "Error!");
            else
            {
                switch (x_cur)
                {
                case 3:
                    studentHead->ScoreBoardCourse.midMark = mark;
                    break;
                case 4:
                    studentHead->ScoreBoardCourse.finalMark = mark;
                    break;
                case 5:
                    studentHead->ScoreBoardCourse.otherMark = mark;
                    break;
                case 6:
                    studentHead->ScoreBoardCourse.totalMark = mark;
                    break;
                default:
                    break;
                }
                stop = true;
            }
        }
        else
            stop = !Message_YesNo("Please re-enter your score. 0 <= mark <= 10\n Press yes to re-enter or no to stop.", "Error!");
    } while (!stop);
    updateSBC(ChooseCourse, x, y, x_cur, y_cur);
}
```

- The “UpdateStudentResult” function allows the user to choose the course, that the user wants to update in the current semester. Then, it will call the updateSBC function.

```
117  void UpdateStudentResult(Year *Yhead)
118  {
119      system("cls");
120      Render_Update(40, 3);
121      Semester *ChooseSem = nullptr;
122      Course *ChooseCourse = nullptr;
123      if (Yhead == nullptr)
124      {
125          Message_Warning("There are no school years at all", "Error not exist!");
126          // gọi lại hàm mainmenu
127          return;
128      }
129      Semester *sem_cur = Yhead->NoSemester;
130      // lấy học kì hiện tại của năm hiện tại
131      if (sem_cur == nullptr)
132      {
133          Message_Warning("There are no semester in current school year.", "Error not exist!");
134          return;
135      }
136      while (sem_cur && sem_cur->next)
137          sem_cur = sem_cur->next;
138      if (sem_cur->course == nullptr)
139      {
140          Message_Warning("There are no course in current semester.", "Error not exist!");
141          return;
142      }
143      goToXY(45, 15);
144      cout << "Select the student's course, in which you want to update scoreboard.";
145      ChooseCourse = chooseCoursebyOption_XY(sem_cur->course, 60, 17, 5);
146      goToXY(45, 15);
147      cout << setw(100) << " ";
148      if (!ChooseCourse)
149          return;
150      int x = 30, y = 17, x_cur = 0, y_cur = 0;
151      goToXY(20, 14);
152      cout << "Select the student for whom you want to update scoreboard.";
153      updateSBC(ChooseCourse, x, y, x_cur, y_cur);
154      UpdateStudentResult(Yhead);
155  }
```

- File: [Course]_View.cpp: This file contains functions that allow the user to choose a choice in the list of the choice and then display it on screen.
    - The “ViewStudentCourse” displays a table containing the list of students in a course with three columns: No, ID, and Student’s Name.

```
void ViewStudentCourse(Course *ChooseCourse, int x, int y)
{
    int width[7];
    width[0] = 10; // chiều rộng cột NO
    width[1] = 15; // chiều rộng cột ID
    width[2] = 35; // chiều rộng Student Name;

    string *title = new string[3];
    title[0] = "No";
    title[1] = "ID";
    title[2] = "Student's Name";

    int num_row = amountStudentOfCourse(ChooseCourse->studentCourse);
    int num_col = 3;

    string **table = new string *[num_row];
    for (int i = 0; i < num_row; i++)
        table[i] = new string[num_col];

    int height = 1, Row_eachTime = 7, Col_eachTime = 8;
    bool edit_Col[3] = {false, false, false};
    StudentCourse *student_cur = ChooseCourse->studentCourse;
    for (int i = 0; i < num_row; i++)
    {
        int j = 0;
        table[i][j++] = to_string(i + 1);
        table[i][j++] = student_cur->ID;
        table[i][j++] = student_cur->FullName;
        student_cur = student_cur->next;
    }
    int x_cur = 0, y_cur = 0;
    Draw_table(table, title, num_row, num_col, width, height, x, y, Row_eachTime, Col_eachTime, edit_Col, x_cur, y_cur);
    if (x_cur == -1)
    {
        for (int i = 0; i < num_row; i++)
            delete[] table[i];
        delete[] table;
        return;
    }
    // này là để chọn được course
    // StudentCourse *curStudent = ChooseCourse->studentCourse;
    // if (!curStudent)
    // {
    //     Message_Warning(" There is no student in course " + ChooseCourse->Name, "Error not exist");
    //     system("cls");
    //     return;
    // }
}
```

- The “ViewScoreBoardCourse” function displays a table containing the list of students’ information and student scoreboard in a course with seven columns: No, ID, Student’s Name, MidMark, FinalMark, OtherMark, and TotalMark.

```
void ViewScoreBoardCourse(Course *ChooseCourse, int x, int y)
{
    int width[7];
    width[0] = 5;  // chiều rộng cột NO
    width[1] = 12; // chiều rộng cột ID
    width[2] = 30; // chiều rộng Name Course;
    width[3] = 15; // chiều rộng Credits
    width[4] = 15; // chiều rộng Days
    width[5] = 15; // chiều rộng Session
    width[6] = 15; // chiều rộng Room

    string *title = new string[8];
    title[0] = "No";
    title[1] = "ID";
    title[2] = "Name Student";
    title[3] = "Mid Mark";
    title[4] = "Final Mark";
    title[5] = "Other Mark";
    title[6] = "Total Mark";

    StudentCourse *student_cur = ChooseCourse->studentCourse;
    int num_row = amountStudentOfCourse(student_cur);
    int num_col = 7;
    string **table = new string *[num_row];
    for (int i = 0; i < num_row; i++)
        table[i] = new string[num_col];

    int height = 1, Row_eachTime = 7, Col_eachTime = 7;
    bool edit_Col[7] = {false, false, false, false, false, false, false};
    for (int i = 0; student_cur != nullptr; i++)
    {
        int j = 0;
        table[i][j++] = to_string(i + 1);
        table[i][j++] = student_cur->ID;
        table[i][j++] = student_cur->FullName;
        table[i][j++] = FormatMark(student_cur->ScoreBoardCourse.midMark);
        table[i][j++] = FormatMark(student_cur->ScoreBoardCourse.finalMark);
        table[i][j++] = FormatMark(student_cur->ScoreBoardCourse.otherMark);
        table[i][j++] = FormatMark(student_cur->ScoreBoardCourse.totalMark);
        student_cur = student_cur->next;
    }
    int x_cur = 0, y_cur = 0;
    Draw_table(table, title, num_row, num_col, width, height, x, y, Row_eachTime, Col_eachTime, edit_Col, x_cur, y_cur);
}
```

- The “ViewCourseOfAStudent” function displays a table containing the list of courses’ information in a semester with columns: No, ID, Course’s Name, Credits, Day, Session, Room, and Teacher Name.

```
void ViewCoursesOfAStudent(Account *accHead, Year *yearHead) // courseHead in a semester, accHead is curent account after the student login
{
    system("cls");
    Render_Course(40, 1);
    Render_Student(35, 7);
    int x = 20, y = 15;

    int width[8];
    width[0] = 5;  // chiều rộng cột NO
    width[1] = 12; // chiều rộng cột ID
    width[2] = 30; // chiều rộng Name Course;
    width[3] = 10; // chiều rộng Credits
    width[4] = 10; // chiều rộng Days
    width[5] = 10; // chiều rộng Session
    width[6] = 10; // chiều rộng Room
    width[7] = 25; // chiều rộng Teacher Name

    string *title = new string[8];
    title[0] = "No";
    title[1] = "ID";
    title[2] = "Name";
    title[3] = "Credits";
    title[4] = "Day";
    title[5] = "Session";
    title[6] = "Room";
    title[7] = "TeacherName ";

    Student *stu_cur = findStudentbyID(accHead->username, yearHead);
    if (stu_cur->course == nullptr)
    {
        Message_Warning("This student is not taking any courses.", "Error not exist");
        return;
    }
    int num_row = amountCourseOfStudent(stu_cur->course);
    int num_col = 8;

    string **table = new string *[num_row];
    for (int i = 0; i < num_row; i++)
        table[i] = new string[num_col];

    CourseStudent *course_cur = stu_cur->course;
    int height = 1, Row_eachTime = 7, Col_eachTime = 8;
    bool edit_Col[8] = {false, false, false, false, false, false, false, false};
    for (int i = 0; i < num_row; i++)
    {
        int j = 0;
        table[i][j++] = to_string(i + 1);
        table[i][j++] = course_cur->course->CourseID;
        table[i][j++] = course_cur->course->Name;
        table[i][j++] = to_string(course_cur->course->Credits);
        table[i][j++] = course_cur->course->Day;
        table[i][j++] = course_cur->course->Session;
        table[i][j++] = course_cur->course->Room;
        table[i][j++] = course_cur->course->TeacherName;
        course_cur = course_cur->next;
    }
    int x_cur = 0, y_cur = 0;
    // bắt đầu in
    goToXY(x, y++);
    cout << "All course of " << accHead->lastName + " " + accHead->firstName << ". MSSV: " << accHead->username;
    Draw_table(table, title, num_row, num_col, width, height, x, y, Row_eachTime, Col_eachTime, edit_Col, x_cur, y_cur);
}
```

vi. [Class] Section

- File: [Class]_Interface.cpp
    - The “Interface_ViewScoreBoardClass” function will let the user choose a class in the system, then select the year, and finally select the semester in that year to display all students’ scoreboards of each course.

```
4  void Interface_ViewStudentClass(Year *yearHead) // task 16: view students in a class
5  {
6      system("cls");
7      int x = 60, y = 17;
8      Render_StudentClass();
9      Class *ChooseClass = nullptr;
10     goToXY(42, y - 2);
11     cout << "Select the Class, in which you want to show list student.";
12     ChooseClass = chooseClass_inAllYear_byOption_XY(yearHead, x, y, 5);
13     goToXY(42, y - 2);
14     cout << setw(100) << " ";
15     if (ChooseClass == nullptr)
16         return;
17
18     if (ChooseClass->StudentClass == nullptr)
19         Message_Warning("There are no student in this class", "Error not Exist");
20     else
21     {
22         goToXY(40, y - 1);
23         cout << " The list of student of " << ChooseClass->Name << " class:";
24         ViewStudentClass(ChooseClass, 40, y);
25     }
26     Interface_ViewStudentClass(yearHead);
27 }
```

- The “Interface_ViewStudentClass” function allows users to choose a class in the system, and then it will display a list of students in the class.

```
void Interface_ViewScoreBoardClass(Year *yearHead)
{
    int x = 60, y = 17;
    system("cls");
    Render_ScoreBoardClass();

    Class *ChooseClass = nullptr;
    Semester *ChooseSem = nullptr;
    Year *ChooseYear_Sem = nullptr;
    Year *year_cur = yearHead;
    while (year_cur)
    {
        if (year_cur->Class)
            break;
        year_cur = year_cur->next;
    }
    if (!year_cur)
    {
        Message_Warning("There is no class in all school year ", "Error");
        return;
    }

    goToXY(45, y - 2);
    cout << "Select the Class, in which you want to show scoreboard.";
    ChooseClass = chooseClass_inAllYear_byOption_XY(yearHead, x, y, 5);
    goToXY(45, y - 2);
    cout << setw(100) << " ";
    while (ChooseClass)
    {
        if (!ChooseClass->StudentClass)
            Message_Warning("There are no student in class " + ChooseClass->Name + ".", "Error not exist");
        else
        {
            goToXY(35, y - 2);
            cout << "Select the school year that contains the semester, in which you want to show list scoreboard";
            ChooseYear_Sem = chooseYearbyOption_XY(yearHead, x, y, 5);
            goToXY(35, y - 2);
            cout << setw(100) << " ";
            while (ChooseYear_Sem)
            {
                if (ChooseYear_Sem->NoSemester == nullptr)
                    Message_Warning("There is no semester in year " + to_string(ChooseYear_Sem->yearStart) + " " + to_string(ChooseYear_Sem->yearStart + 1), "Error");
                else
                {
                    goToXY(45, y - 2);
                    cout << "Select the Semester, in which you want to show scoreboard.";
                    ChooseSem = chooseSemesterbyOption_XY(ChooseYear_Sem->NoSemester, x, y, 5);
                    goToXY(45, y - 2);
                    cout << setw(100) << " ";
                    while (ChooseSem)
                    {
                        ViewScoreboardClass(ChooseClass, ChooseSem, 20, y);
                        system("cls");
                        Render_ScoreBoardClass();
                        goToXY(45, y - 2);
                        cout << "Select the Semester, in which you want to show scoreboard.";
                        ChooseSem = chooseSemesterbyOption_XY(ChooseYear_Sem->NoSemester, x, y, 5);
                        goToXY(45, y - 2);
                        cout << setw(100) << " ";
                    }
                }
                goToXY(35, y - 2);
                cout << "Select the school year that contains the semester, in which you want to show list scoreboard";
                ChooseYear_Sem = chooseYearbyOption_XY(yearHead, x, y, 5);
                goToXY(35, y - 2);
                cout << setw(100) << " ";
            }
        }
        goToXY(45, y - 2);
        cout << "Select the Class, in which you want to show scoreboard.";
        ChooseClass = chooseClass_inAllYear_byOption_XY(yearHead, x, y, 5);
        goToXY(45, y - 2);
        cout << setw(100) << " ";
    }
}
```

- File: [Class]_View.cpp: This file contains functions that allow the user to choose a choice in the list of the choice and then display it on screen.

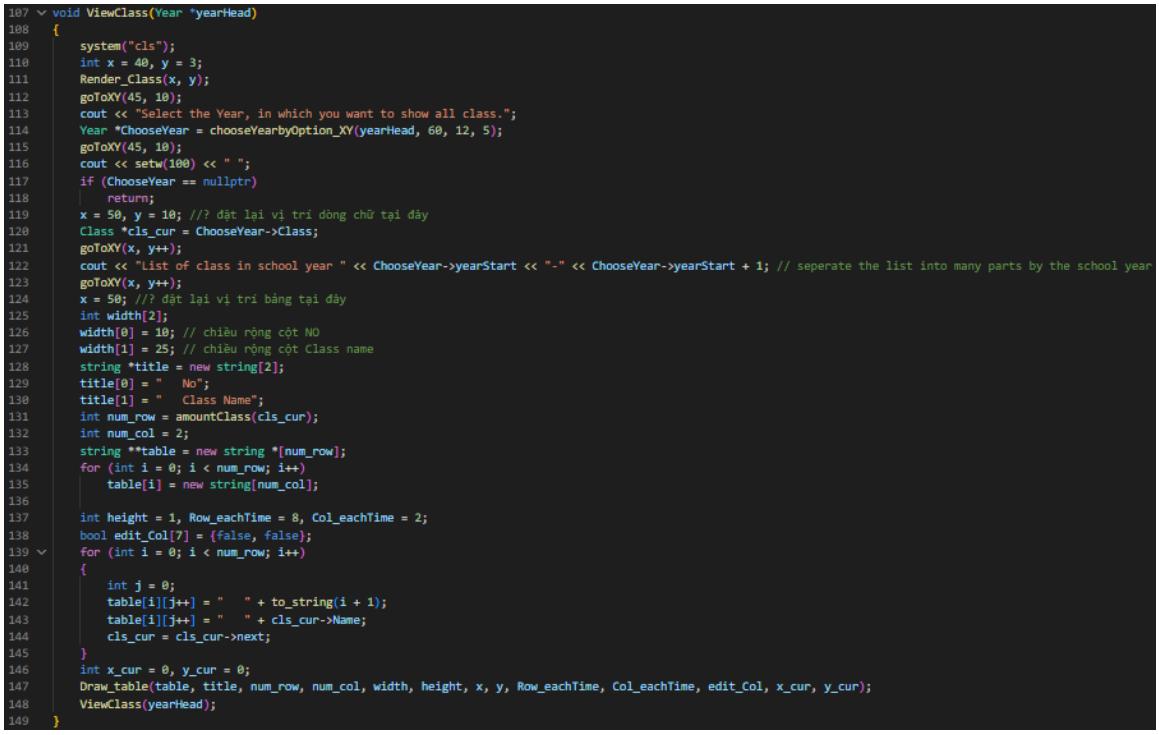
- The “ViewStudentClass” displays a table containing the list of students in a class with three columns: No, ID, and Student’s Name.

```
void ViewStudentClass(Class *ChooseClass, int x, int y)
{
    Student *student_cur = ChooseClass->StudentClass;
    goToXY(x, y++);
    cout << "List of students in " << ChooseClass->Name;
    int width[3];
    width[0] = 10; // chiều rộng cột NO
    width[1] = 15; // chiều rộng cột ID
    width[2] = 35; // chiều rộng Student Name;

    string *title = new string[3];
    title[0] = "No";
    title[1] = "ID";
    title[2] = "Student Name";

    int num_row = amountStudentOfClass(ChooseClass->StudentClass);
    int num_col = 3;

    string **table = new string *[num_row];
    for (int i = 0; i < num_row; i++)
        table[i] = new string[num_col];

    int height = 1, Row_eachTime = 6, Col_eachTime = 8;
    bool edit_Col[3] = {false, false, false};

    for (int i = 0; student_cur != nullptr; i++)
    {
        int j = 0;
        table[i][j++] = to_string(i + 1);
        table[i][j++] = student_cur->ID;
        table[i][j++] = student_cur->accStudent->lastName + " " + student_cur->accStudent->firstName;
        student_cur = student_cur->next;
    }
    int x_cur = 0, y_cur = 0;
    Draw_table(table, title, num_row, num_col, width, height, x, y, Row_eachTime, Col_eachTime, edit_Col, x_cur, y_cur);
}
```

- The “ViewScoreBoardClass” function displays a table containing the list of students’ information and student scoreboard in a class with eight columns: No, ID, Cousre’s Name, MidMark, FinalMark, OtherMark, TotalMark, and GPA.

```
41  void ViewScoreboardClass(Class *Class, Semester *ChooseSem, int x, int y)
42  {
43      Render_ScoreBoardClass();
44      Student *student_cur = Class->StudentClass;
45      CourseStudent *courseHead = CourseOfAClass_InChooseSem(student_cur, ChooseSem);
46      if (courseHead == nullptr)
47      {
48          Message_Warning(Class->Name + " class has no subjects this semester", "Error not exist");
49          return;
50      }
51
52      int num_col = amountCourseOfStudent(courseHead);
53      int num_row = amountStudentOfClass(student_cur);
54      double **SBC;
55      createSBC_ofClass(SBC, courseHead, student_cur, num_col, num_row);
56      num_col += 4;
57
58      string *title = new string[num_col];
59      CourseStudent *course_cur = courseHead;
60      title[0] = "No";
61      title[1] = "ID";
62      title[2] = "Full Name";
63      for (int i = 3; course_cur != nullptr; i++, course_cur = course_cur->next)
64      {
65          title[i] = course_cur->course->Name;
66      }
67      title[num_col - 1] = "GPA";
68      string **table = new string *[num_row];
69      for (int i = 0; i < num_row; i++)
70          table[i] = new string[num_col];
71
72      int height = 1, Row_eachTime = 7, Col_eachTime = 7;
73      bool *edit_Col = new bool[num_col];
74      for (int i = 0; i < num_col; i++)
75          edit_Col[i] = false;
76      for (int i = 0; student_cur != nullptr; i++)
77      {
78          int j = 0;
79          table[i][j++] = to_string(i + 1);
80          table[i][j++] = student_cur->ID;
81          table[i][j++] = student_cur->accStudent->lastName + " " + student_cur->accStudent->firstName;
82          while (j < num_col)
83          {
84              table[i][j] = FormatMark(SBC[i][j - 3]);
85              j++;
86          }
87          student_cur = student_cur->next;
88      }
89      int *width = new int[num_col];
90      width[0] = 5;  // chiều rộng cột NO
91      width[1] = 12; // chiều rộng cột ID
92      width[2] = 30; // chiều rộng Student Name;
93      for (int i = 3; i < num_col - 1; i++)
94          width[i] = title[i].length() + 2;
95      width[num_col - 1] = 10;
96      int x_cur = 0, y_cur = 0;
97      Draw_table(table, title, num_row, num_col, width, height, x, y, Row_eachTime, Col_eachTime, edit_Col, x_cur, y_cur);
98
99      for (int i = 0; i < num_row; i++)
100         delete[] SBC[i];
101     delete[] SBC;
102     delete[] title;
103     delete[] width;
104     delete[] edit_Col;
105 }
```

- The “ViewClass” function displays a table containing the list of class information in a year, that uses selected with two columns: No, Class Name.



```
void ViewClass(Year *yearHead)
{
    system("cls");
    int x = 40, y = 3;
    Render_Class(x, y);
    goToXY(45, 10);
    cout << "Select the Year, in which you want to show all class.";
    Year *ChooseYear = chooseYearbyOption_XY(yearHead, 60, 12, 5);
    goToXY(45, 10);
    cout << setw(100) << " ";
    if (ChooseYear == nullptr)
        return;
    x = 50, y = 10; //? đặt lại vị trí dòng chữ tại đây
    Class *cls_cur = ChooseYear->Class;
    goToXY(x, y++);
    cout << "List of class in school year " << ChooseYear->yearStart << "-" << ChooseYear->yearStart + 1; // seperate the list into many parts by the school year
    goToXY(x, y++);
    x = 50; //? đặt lại vị trí bảng tại đây
    int width[2];
    width[0] = 10; // chiều rộng cột NO
    width[1] = 25; // chiều rộng cột Class name
    string *title = new string[2];
    title[0] = "   No";
    title[1] = "   Class Name";
    int num_row = amountClass(cls_cur);
    int num_col = 2;
    string **table = new string *[num_row];
    for (int i = 0; i < num_row; i++)
        table[i] = new string[num_col];

    int height = 1, Row_eachTime = 8, Col_eachTime = 2;
    bool edit_Col[7] = {false, false};
    for (int i = 0; i < num_row; i++)
    {
        int j = 0;
        table[i][j++] = "   " + to_string(i + 1);
        table[i][j++] = "   " + cls_cur->Name;
        cls_cur = cls_cur->next;
    }
    int x_cur = 0, y_cur = 0;
    Draw_table(table, title, num_row, num_col, width, height, x, y, Row_eachTime, Col_eachTime, edit_Col, x_cur, y_cur);
    ViewClass(yearHead);
}
```

vii. [Student] Section

- File [Student]_Interface.cpp: This file contains student functions that allow the student to choose a choice in the list of the choice and then display it on screen.
    - The “Interface_ViewCoursesOfUser” function will allow the user to choose a year school or back to the student main menu, then select a semester in that year, and finish it will display all the courses in the semester.

```
void Interface_ViewCoursesOfUser(Student *student_cur, Year *yearHead)
{
    if (student_cur->course == nullptr)
    {
        Message_Warning("This student is not taking any courses.", "Error not exist");
        return;
    }
    if (yearHead == nullptr)
    {
        Message_Warning("There are no school year.", "Error not exist");
        // gọi lại hàm mainmenu
        return;
    }
    int x = 60, y = 15;
    Year *ChooseYear = nullptr;
    Semester *ChooseSem = nullptr;
    if (yearHead == nullptr)
    {
        Message_Warning("There are no school year.", "Error not exist");
        // gọi lại hàm mainmenu
        return;
    }
    goToXY(40, 13);
    cout << "Select the year that contains the semester you want to show all course";
    ChooseYear = chooseYearbyOption_XY(yearHead, x, y, 5);
    cout << setw(70) << " ";
    if (ChooseYear == nullptr)
        // quay lại main menu
        return;
    if (ChooseYear->NoSemester == nullptr)
    {
        Message_Warning("There are no semester in this school year.", "Error not exist");
        Interface_ViewCoursesOfUser(student_cur, yearHead);
        return;
    }
    goToXY(40, 13);
    cout << "Select the semester that contains the course you want to show all course";
    ChooseSem = chooseSemesterbyOption_XY(ChooseYear->NoSemester, x, y, 5);
    cout << setw(70) << " ";
    while (ChooseSem)
    {
        ViewCoursesOfUser(student_cur, ChooseSem->course);
        system("cls");
        Render_Course(40, 1);
        Render_Student(35, 7);
        goToXY(40, 13);
        cout << "Select the semester that contains the course you want to show all course";
        ChooseSem = chooseSemesterbyOption_XY(ChooseYear->NoSemester, x, y, 5);
    }
    Interface_ViewCoursesOfUser(student_cur, yearHead);
}
```

- The “Interface_ViewScoreBoardOfUser” function will allow the user to choose a year school or back to the student main menu, then select a semester in that year, and finish it will display the scoreboard of that student in all the courses of the semester.

```cpp
void Interface_ViewScoreBoardOfUser(Student *student_cur, Year *yearHead) //? task 14
{
    Year *ChooseYear = nullptr;
    Semester *ChooseSem = nullptr;
    system("cls");
    int x = 20, y = 2;
    Render_ScoreBoard(x, y);
    x = 60, y = 15;
    if (yearHead == nullptr)
    {
        Message_Warning("There are no school year.", "Error not exist");
        // gọi lại hàm mainmenu
        return;
    }
    goToXY(40, 13);
    cout << "Select the year that contains the semester you want to show scoreboard";
    ChooseYear = chooseYearbyOption_XY(yearHead, x, y, 5);
    cout << setw(70) << " ";
    if (ChooseYear == nullptr)
        // quay lại main menu
        return;
    if (ChooseYear->NoSemester == nullptr)
    {
        Message_Warning("There are no semester in this school year.", "Error not exist");
        Interface_ViewScoreBoardOfUser(student_cur, yearHead);
        return;
    }
    goToXY(40, 13);
    cout << "Select the semester that contains the course you want to show scoreboard";
    ChooseSem = chooseSemesterbyOption_XY(ChooseYear->NoSemester, x, y, 5);
    cout << setw(70) << " ";
    while (ChooseSem)
    {
        if (ChooseSem->course == nullptr)
            Message_Warning("There are no course in this semester", "Error not Exist");
        else
            ViewScoreboard(student_cur->accStudent, ChooseSem->course);
        system("cls");
        Render_ScoreBoard(20, 2);
        goToXY(40, 13);
        cout << "Select the semester that contains the course you want to show scoreboard";
        ChooseSem = chooseSemesterbyOption_XY(ChooseYear->NoSemester, x, y, 5);
        cout << setw(70) << " ";
    }
    Interface_ViewScoreBoardOfUser(student_cur, yearHead);
}
```

- File: [Students]_View.cpp: This file contains view functions that allow the user to choose a choice in the list of the choice and then display it on screen.
    - The “ViewCourseOfUser” function displays a table containing the list of courses’ information in a semester with columns: No, ID, Course’s Name, Credits, Day, Session, Room, and Teacher Name.

```
void ViewCoursesOfUser(Student *student_cur, Course *courseHead) // courseHead in a semester, accHead is curent account after the student login
{
    system("cls");
    Render_Course(40, 1);
    Render_Student(35, 7);
    int x = 20, y = 15;

    int width[8];
    width[0] = 5;  // chiều rộng cột NO
    width[1] = 12; // chiều rộng cột ID
    width[2] = 30; // chiều rộng Name Course;
    width[3] = 10; // chiều rộng Credits
    width[4] = 10; // chiều rộng Days
    width[5] = 10; // chiều rộng Session
    width[6] = 10; // chiều rộng Room
    width[7] = 25; // chiều rộng Teacher Name

    string *title = new string[8];
    title[0] = "No";
    title[1] = "ID";
    title[2] = "Name";
    title[3] = "Credits";
    title[4] = "Day";
    title[5] = "Session";
    title[6] = "Room";
    title[7] = "TeacherName ";

    int num_row = 0;
    CourseStudent *listcourse = nullptr;
    while (courseHead)
    {
        CourseStudent *course_cur = student_cur->course;
        while (course_cur)
        {
            if (course_cur->course == courseHead)
            {
                CourseStudent *tmp = new CourseStudent;
                tmp->next = listcourse;
                tmp->course = course_cur->course;
                listcourse = tmp;
                num_row++;
            }
            course_cur = course_cur->next;
        }
        courseHead = courseHead->next;
    }
    if (listcourse == nullptr)
    {
        Message_Warning("This student is not taking any courses in this Semester.", "Error not exist");
        return;
    }
    int num_col = 8;

    string **table = new string *[num_row];
    for (int i = 0; i < num_row; i++)
        table[i] = new string[num_col];

    CourseStudent *course_cur = listcourse;
    int height = 1, Row_eachTime = 8, Col_eachTime = 8;
    bool edit_Col[8] = {false, false, false, false, false, false, false, false};
    for (int i = 0; i < num_row; i++)
    {
        int j = 0;
        table[i][j++] = to_string(i + 1);
        table[i][j++] = course_cur->course->CourseID;
        table[i][j++] = course_cur->course->Name;
        table[i][j++] = to_string(course_cur->course->Credits);
        table[i][j++] = course_cur->course->Day;
        table[i][j++] = course_cur->course->Session;
        table[i][j++] = course_cur->course->Room;
        table[i][j++] = course_cur->course->TeacherName;
        course_cur = course_cur->next;
    }
    int x_cur = 0, y_cur = 0;
    // bắt đầu in
    goToXY(x, y++);
    cout << "All course of " << student_cur->accStudent->lastName + " " + student_cur->accStudent->firstName << ". MSSV: " << student_cur->accStudent->username;
    Draw_table(table, title, num_row, num_col, width, height, x, y, Row_eachTime, Col_eachTime, edit_Col, x_cur, y_cur);
}
```

- The “ViewScoreBoard” function displays a table containing the list of students’ information and student scoreboard in a course with seven columns: No, ID, Course’s Name, MidMark, FinalMark, OtherMark, and TotalMark.

```
void ViewScoreboard(Account *accHead, Course *courseHead)
{
    system("cls");
    Render_ScoreBoard(25, 1); // in ra chữ scoreboard bắt đầu từ ngang 25 dọc 1

    int width[7];  // chiều rộng của các cột
    width[0] = 8;  // chiều rộng cột NO
    width[1] = 14; // chiều rộng cột ID
    width[2] = 30; // chiều rộng Name Course;
    width[3] = 15; // chiều rộng Mid
    width[4] = 15; // chiều rộng Final
    width[5] = 15; // chiều rộng Other
    width[6] = 15; // chiều rộng Total

    string *title = new string[8];
    title[0] = "No";
    title[1] = "ID";
    title[2] = "Subject";
    title[3] = "Mid Mark";
    title[4] = "Final Mark";
    title[5] = "Other Mark";
    title[6] = "Total Mark";

    int num_row = 0;
    int num_col = 7;
    CourseStudent *list_course_of_student = nullptr; // danh sách khóa học của học sinh này trong học kì này
    CourseStudent *tmp = nullptr;
    StudentCourse *cur_stu = nullptr;
    // đếm số course của học sinh trong kì này và tạo list course mà student này học trong kì này
    while (courseHead)
    {
        cur_stu = courseHead->studentCourse;
        while (cur_stu)
        {

            if (cur_stu->ID == accHead->username) // truy cập từng student trong từng course check ID
            {
                num_row++;
                if (!list_course_of_student)
```

```
                    list_course_of_student = new CourseStudent;
                else
                {
                    tmp = new CourseStudent;
                    tmp->next = list_course_of_student;
                    list_course_of_student = tmp;
                }
                list_course_of_student->course = courseHead;
            }
            cur_stu = cur_stu->next;
        }
        courseHead = courseHead->next;
    }
    if (list_course_of_student == nullptr)
    {
        Message_Warning("This student is not taking any courses in this semester.", "Error not exist");
        return;
    }
    int height = 1, Row_eachTime = 8, Col_eachTime = 7;
    bool edit_Col[7] = {false, false, false, false, false, false, false}; // không có cột nào được sửa nhen
    tmp = list_course_of_student;
    // tạo table điểm để in ra ở draw table
    string **table = new string *[num_row];
    for (int i = 0; i < num_row; i++)
        table[i] = new string[num_col];
    for (int i = 0; tmp != nullptr; i++)
    {
        int j = 0;
        table[i][j++] = to_string(i + 1); // No
        table[i][j++] = tmp->course->CourseID;
        table[i][j++] = tmp->course->Name;
        cur_stu = find_StudentOfCourse(accHead->username, tmp->course->studentCourse);
        table[i][j++] = FormatMark(cur_stu->ScoreBoardCourse.midMark);
        table[i][j++] = FormatMark(cur_stu->ScoreBoardCourse.finalMark);
        table[i][j++] = FormatMark(cur_stu->ScoreBoardCourse.otherMark);
        table[i][j++] = FormatMark(cur_stu->ScoreBoardCourse.totalMark);
        tmp = tmp->next;
    }
    int x_cur = 0, y_cur = 0;
    int x = 25, y = 8;
    goToXY(x, y++);
    cout << "ScoreBoard of " << accHead->lastName + " " + accHead->firstName << ". MSSV: " << accHead->username;
    goToXY(x, y++);

    Draw_table(table, title, num_row, num_col, width, height, x, y, Row_eachTime, Col_eachTime, edit_Col, x_cur, y_cur);
    // dưới này là delete linked list và mảng động
    while (list_course_of_student)
    {
        tmp = list_course_of_student;
        list_course_of_student = list_course_of_student->next;
        delete tmp;
    }
    for (int i = 0; i < num_row; i++)
        delete[] table[i];
    delete[] table;
}
```

viii. Login.cpp

- "LoggingIn" prints out the boxes for users to enter the username and password and checks through the account linked list to verify it to allow users to access in the program.

```
5  void LoggingIn(Account *accHead, std::string &user, std::string &pass, int &role)
6  {
7      system("cls");
8      Render_Login();
9
10     goToXY(50, 18);
11     TextColor(0x0E);
12     cout << "Username";
13
14     TextColor(63);
15     for (int i = 0; i < 3; i++)
16     {
17         goToXY(50, 19 + i);
18         cout << "                                                  ";
19     }
20
21     goToXY(50, 23);
22     TextColor(0x0E);
23     cout << "Password";
24     TextColor(63);
25     for (int i = 0; i < 3; i++)
26     {
27         goToXY(50, 24 + i);
28         cout << "                                                  ";
29     }
30
31     user = Limit_Input(52, 20, 30, 63);
32     pass = Limit_Input(52, 25, 30, 63);
33     TextColor(WHITE);
34
35     Account *cur = accHead;
36     bool loggedIn = false;
37     while (cur)
38     {
39         if (cur->username == user && cur->password == pass)
40         {
41             loggedIn = true;
42             break;
43         }
```

```
        cur = cur->next;
    }

    if (!loggedIn)
    {
        string message = "Wrong username or password!!!\nRetry!";
        string title = "Login Failed";
        Message_Warning(message, title);
        // TextColor(LIGHT_RED);

        // goToXY(60, 28);
        // cout << "Wrong username or password";
        // goToXY(66, 29);

        // cout << "Try again!!!";
        // TextColor(WHITE);

        // Pause();
        LoggingIn(accHead, user, pass, role);
        return;
    }
    role = cur->role;
    string message = "Logged in successfully!!\nHello, " + cur->lastName + " " + cur->firstName + "!";
    string title = "Login Success";
    Message_Warning(message, title);
    // TextColor(LIGHT_YELLOW);
    // goToXY(60, 28);
    // cout << "Logged in successfully!!";
    // goToXY(60, 29);
    // cout << "Hello, " << cur->lastName << ' ' << cur->firstName << "!";
    // Pause();

    return;
}
```

- "ChangePass" makes users re-enter their password to change into the new one and save it into the data bank.

```
void ChangePass(Account *accHead, std::string &user, std::string &pass)
{
    system("cls");
    Render_Account(50, 1);
    TextColor(0x0B);
    goToXY(50, 18);
    TextColor(0x0E);
    cout << "Old Password";
    TextColor(63);
    for (int i = 0; i < 3; i++)
    {
        goToXY(50, 19 + i);
        cout << "                                        ";
    }
    goToXY(50, 23);
    TextColor(0x0E);
    cout << "New Password";
    TextColor(63);
    for (int i = 0; i < 3; i++)
    {
        goToXY(50, 24 + i);
        cout << "                                        ";
    }
    goToXY(50, 28);
    TextColor(0x0E);
    cout << "Confirm Password";
    TextColor(63);
    for (int i = 0; i < 3; i++)
    {
        goToXY(50, 29 + i);
        cout << "                                        ";
    }
    string tmp;
    TextColor(63);
    tmp = Limit_Input(52, 20, 30, 63);

    while (tmp != pass) // If the password is wrong
    {
        TextColor(0x3C);
```

```
118         goToXY(80, 20);
119         cout << "Wrong Password";
120         Pause();
121         TextColor(63);
122         goToXY(50, 20);
123         cout << "                                                  ";
124         tmp = Limit_Input(52, 20, 30, 63);
125     }
126     string p1, p2;
127     p1 = Limit_Input(52, 25, 30, 63);
128     p2 = Limit_Input(52, 30, 30, 63);
129     while (p1 != p2)
130     {
131         string message = "The new password and the confirm password are not the same!!!\nPlease try again!!!";
132         string title = "Change Password Failed";
133         if (!Message_YesNo(message, title))
134             return;
135         TextColor(63);
136         goToXY(50, 25);
137         cout << "                                                  ";
138         goToXY(50, 30);
139         cout << "                                                  ";
140         p1 = Limit_Input(52, 25, 30, 63);
141         p2 = Limit_Input(52, 30, 30, 63);
142     }
143     Account *cur = accHead;
144     while (cur && cur->username != user) // Find the account
145         cur = cur->next;
146     pass = cur->password = p1;
147     WriteAccount(accHead);
148     string message = "Change password successfully!!!";
149     string title = "Change Password Success";
150     Message_Warning(message, title);
151     TextColor(7);
152     return;
153 }
```

- "Profiling" displays the user's profile including username, password, role, full name, birthday, gender, and social ID.

```
161  void Profiling(Account *accHead, string user, string pass, int role)
162  {
163      // Name, birth, gender, social ID, username, password = ********, role
164      system("cls");
165      Render_Account(50, 1);
166      Account *cur = accHead;
167      while (cur && (cur->username != user || cur->password != pass))
168          cur = cur->next;
169
170      TextColor(63);
171      for (int i = 0; i < 3; i++)
172      {
173          goToXY(50, 9 + i);
174          cout << "                                                    "; // username
175          goToXY(50, 13 + i);
176          cout << "                                    "; // password
177          goToXY(85, 13 + i);
178          cout << "                  "; // Role
179          goToXY(50, 17 + i);
180          cout << "                                                    "; // Full name
181          goToXY(50, 21 + i);
182          cout << "                                    "; // Birthday
183          goToXY(85, 21 + i);
184          cout << "                  "; // Gender
185          goToXY(50, 25 + i);
186          cout << "                                                    "; // Social ID
187      }
188      goToXY(52, 10);
189      cout << "Username: " << cur->username;
190
191      goToXY(52, 14);
192      cout << "Password: " << cur->password.substr(0, 2);
193      for (int i = 2; i < cur->password.length(); i++)
194          cout << '*';
195      goToXY(88, 14);
196      cout << "Role: ";
197      if (cur->role == 1)
198          cout << "Student";
199      else
200          cout << "Staff";
201
202      goToXY(52, 18);
203      cout << "Full name: " << cur->lastName << ' ' << cur->firstName;
204
205      goToXY(52, 22);
206      cout << "Birthday: " << cur->birth;
207
208      goToXY(88, 22);
209      cout << "Gender: " << (cur->Gender == "M" ? "Male" : "Female");
210      goToXY(52, 26);
211      cout << "Social ID: " << cur->SocialID;
212
213      TextColor(7);
214      Pause();
215  }
```

- "AccountAlteration" prints out the menu of managing accounts and calls the corresponding functions to do the tasks.

```
bool AccountAlteration(Year *yearHead, Account *accHead, std::string &user, std::string &pass, int &role)
{
    system("cls");
    Render_Account(50, 1);

    vector<string> menu;
    menu.push_back("Main Menu");
    menu.push_back("Profile");
    menu.push_back("Change password");
    menu.push_back("Logout");
    menu.push_back("Quit");

    int opt = Draw_XY(menu, 60, 12, 5, 20, 63);

    switch (opt)
    {
    case 0:
        return false;

    case 1:
        Profiling(accHead, user, pass, role);
        return AccountAlteration(yearHead, accHead, user, pass, role);

    case 2:
        ChangePass(accHead, user, pass);
        return AccountAlteration(yearHead, accHead, user, pass, role);

    case 3:
        goToXY(50, 28);
        cout << "Returning to login page";
        for (int i = 0; i < 8; i++)
        {
            cout << ".";
            Sleep(200);
        }
        // system("cls");
        if (Message_YesNo("Do you want to save all the changes?", "Notice!"))
        {
            Output(yearHead);  // write down all the year
            Outyear(yearHead); // write down each year in4
            Message_Warning("File save!\nNow clean up and close!", "Quit");
        }
        LoggingIn(accHead, user, pass, role);
        return false;
        // return AccountAlteration(accHead, user, pass, role);
    case 4:
        goToXY(50, 28);
        cout << "Thanks for ur usage!";
        Pause();
        return true;
    }
    return false;
}
```

- "LoggingMain" stores the two functions to minimize the calling function in the other functions to login and manage accounts.

```
bool LoggingMain(Year *yearHead, Account *&accHead, string &user, string &pass, int &role)
{
    // ReadAccount(accHead);
    LoggingIn(accHead, user, pass, role);
    if (AccountAlteration(yearHead, accHead, user, pass, role))
        return true;
    return false;
}
```

ix. Main.cpp

- "EradicateLL" is managed to call all functions that delete the linked list and free up the memory.

```
7  void EradicateLL(Account *&accHead, Year *&yearHead)
8  {
9      DelAccount(accHead);
10     DeleteYear(yearHead);
11 }
```

- "Merge_year_sem" merges all the tasks including recover back the files, reading the accounts and years, linking the information around the linked lists, and calling the main menu that fits the user's role.

```
12 void Merge_year_sem(Account *&accHead, Year *&yearHead)
13 {
14     yearHead = RecoverFile();
15     string user = "", pass = "";
16     int role;
17     ReadAccount(accHead);
18     SyncInForStudent(yearHead, accHead);
19     SyncFullName(yearHead, accHead);
20     SyncCourse(yearHead);
21     LoggingIn(accHead, user, pass, role);
22     Main_Menu(yearHead, accHead, user, pass, role);
23 }
```

4. Summary

In conclusion, the university management software is a comprehensive solution for managing various aspects of a university, including students, courses, and semesters. The software is designed to provide ease of use, scalability, and flexibility to accommodate the ever-changing needs of a university.

With the help of this software, university administrators can easily manage student enrollment, track course attendance, manage student records, and generate various reports. Additionally, the software provides a user-friendly interface for students to register for courses, view their schedules, and access course materials.

Furthermore, the software is designed to be modular and easily extendable, allowing for the addition of new features and integration with other systems as required. The software has been developed using industry-standard programming languages and technologies, ensuring its reliability and security.

Overall, the university management software provides an all-in-one solution for managing a university, helping to streamline administrative processes

and improve student outcomes. The software is an essential tool for any university looking to improve its management systems and stay competitive in the ever-evolving education industry.

END