VIETNAM NATIONAL UNIVERSITY HO CHI MINH

UNIVERSITY OF SCIENCE

---o0o---

PROGRAMMING TECHNIQUE – CSC10002

PROJECT REPORT

Trần Đăng Tuấn (22127438)
Nguyễn Tấn Châu (22127043)
Trần Nguyễn Minh Hoàng (22127131)
Đoàn Đặng Phương Nam (22127280)

GROUP 4 – 22CLC02

Table of content

Abstract

A School Management System (SMS) is a computer program that helps schools with their day-to-day tasks. It helps with things like keeping records, managing student enrolment, taking attendance, creating schedules, and giving grades. The program has a central database that authorized people can use to manage student information and generate reports easily.

SMSs are really important for schools because they make things easier, save time, and resources. They let staff members automate repetitive tasks and spend more time teaching and researching. SMSs also help students access their course details, grades, and other important information. Lastly, they help schools follow rules and regulations set by educational governing bodies.

In this report, we present a C++ program that implements an SMS to manage a school system. The program provides a comprehensive set of features that allow staff members to perform administrative tasks efficiently and students to access their course information and grades easily. The program uses mainly singly linked lists to store and manage data, making it easy to scale and maintain.

We welcome feedback and constructive criticism from our teachers who will be grading our project. This will help us to identify any areas where improvements can be made and allow us to refine our system. Please contact us via email: ddpnam22@clc.fitus.edu.vn, we are very open to suggestions to improve our program.

Group of authors

I. Struct

First of all, our **Data Structure** is depend on our Subject. These structs define a complex data structure for managing a student information system. The system appears to be organized around the concepts of **years, semesters, classes, students, and courses**, with each struct representing a distinct entity within the system. Here is a brief description of each struct

1. Date

This struct represents a date, with fields for the **day, month, and year**.

```
struct Date
{
    unsigned int day, month, year;
};
```

2. Class

This struct represents a class, with fields for the **class name**, **ID**, a pointer to the head of the linked list of **students** enrolled in the class, and a pointer to the **next class** in the linked list.

```
struct Class
{
    string class_name;
    string class_id;
    Student *student_head; // Points to the Student
    Class *next_class; // Next class
};
```

3. Student

This struct represents a student, with fields for the student ID, social ID, first name, last name, a pointer to the class the student is enrolled in, gender, date of birth, a pointer to the head of the linked list of courses the student is taking, and a pointer to the next student in the linked list.

```
struct Student
{
    string student_ID;
    string student_socialID;
    string student_fisrtname;
    string student_lastname;
    Class student_class;
    bool gender; //0: Male, 1: Female
    Date DOB; // Date of Birth
    Course *course_head; // Points to the Course
    Student *student_next; // Next student
};
```

4. Course

This struct represents a course, with fields for the teacher name, course ID, course name, class name, number of credits, maximum number of students, day of the week the course meets, session (e.g. morning or afternoon), a pointer to the head of the linked list of students enrolled in the course, a pointer to the head of the linked list of scoreboards for the course, and a pointer to the next course in the linked list.

```
struct Course
{
    string teacher_name;
    string course_ID;
    string course_name;
    string class_name;
    int numCredits;
    int maxNumStudents; //default 50
    string dayInWeek;
    string Session;
    Scoreboard *scoreboard_head; //Points to the ScoreBoard
    Course *course_next; //Next Course
    Student *student_head; //Points to the student
};
```

5. Year

This struct represents a school year, with fields for the year name, a pointer to the head of the linked list of classes for the year, a pointer to the head of the linked list of semesters for the year, and a pointer to the next year in the linked list.

```
struct Year
{
    string year_name;
    Class *class_head; //Points to the Class
    Semester *semester_head; //Points to the Semester
    Year *year_next; //Next Year
};
```

6. Semester

This struct represents a semester, with fields for the semester ordinal number, start and end dates, a pointer to the head of the linked list of courses for the semester, and a pointer to the next semester in the linked list.

```
struct Semester
{
    unsigned int Semester_Ord; //Ordinal number
    Date start_date;
    Date end_date;
    Course *course_head; //Points to the Course
    Semester *semester_next; //Next Semester
};
```

Overall, these structs provide a robust and flexible framework for managing a student information system with support for classes, students, courses, semesters, and years. The use of linked lists allows for dynamic creation and modification of the data structure, which is essential for any system dealing with a large and constantly evolving set of data.

II. Structure of source codes

In an effort to summarize the source codes comprehensively, we will be following the structures designed in our interface.

For student data in the classroom, the system will automatically retrieve data from the student_info.csv file every time a STAFF member logs into the system.

1. First Screen

int firstScreen(bool &check): This function is used to display a menu screen for a program. The menu has three options: **"STAFF", "STUDENT" and "OUT PROGRAM".** The user can select an option using the arrow keys on their keyboard, and the selected option will be highlighted. When the user presses **Enter**, the function will return an integer value representing the option selected.

void Myinterface(int option, Year* &year_head): This is a function that takes two arguments: an option and a pointer to a **Year** object. It prompts the user to login and redirects them to either the **staff or student screen** depending on the option chosen. If the login is successful, it clears the console and calls the corresponding screen function.

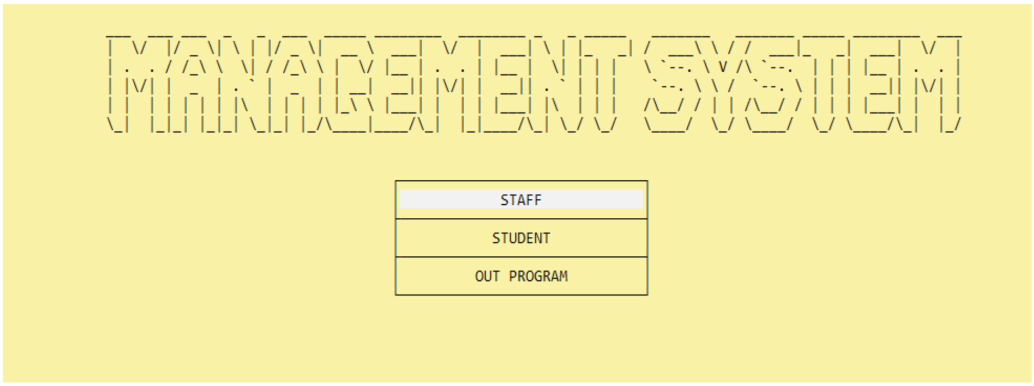


Figure II.1.1: Our first screen

2. General Features

a. Change Password

void changePass(string username, bool isStudent, int& opt): This function encompasses the ability for both staffs as well as students enrolled in the system to **modify their existing passwords**. The user interface is designed in a way that is intended to be **intuitive and user-friendly**, facilitating a seamless and efficient process for users to update their login credentials.

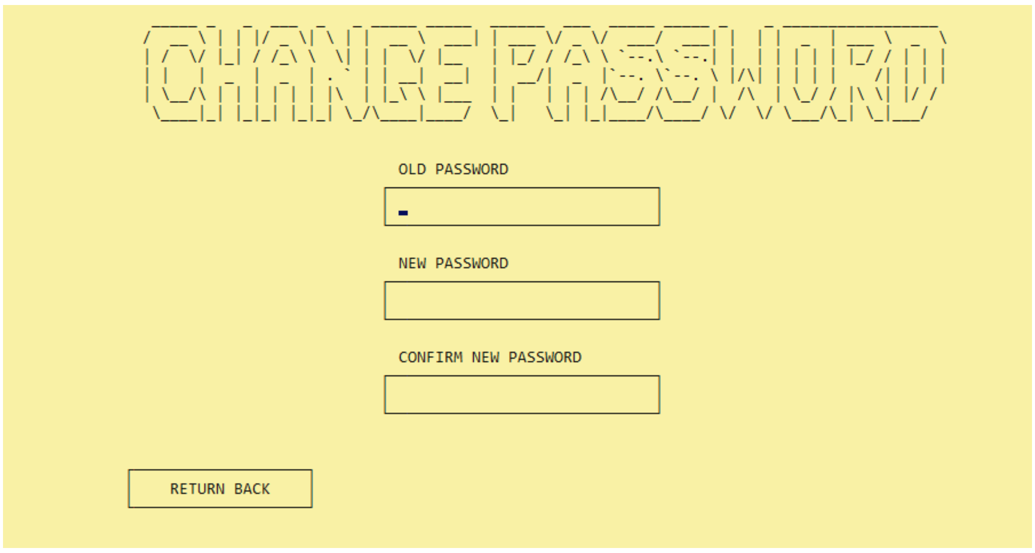


Figure II.2.a.1: ChangePassword Screen

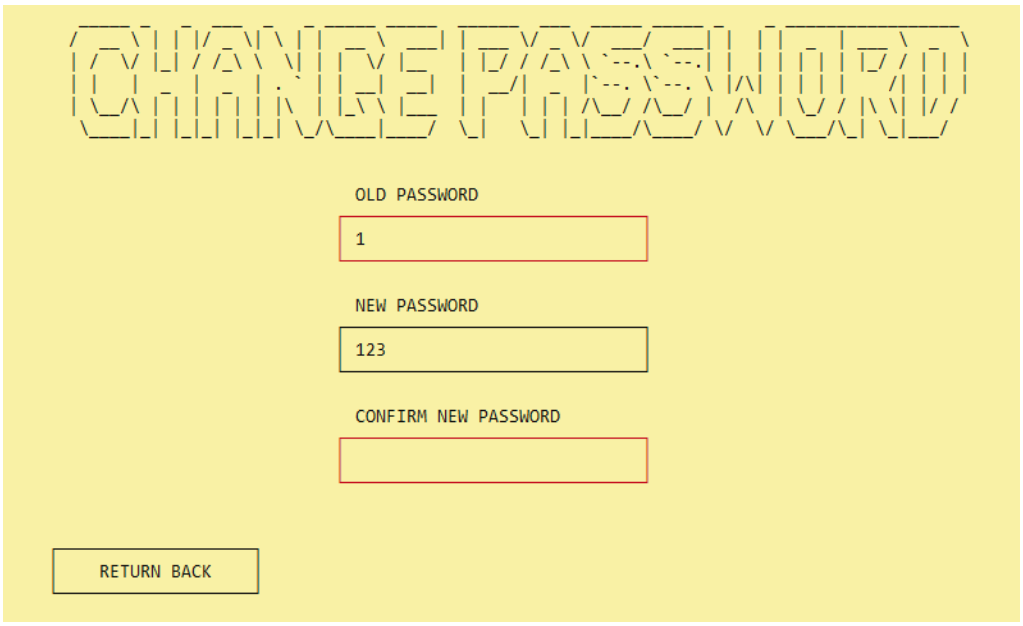


Figure II.2.a.2: The boxes modify color when the information is wrong

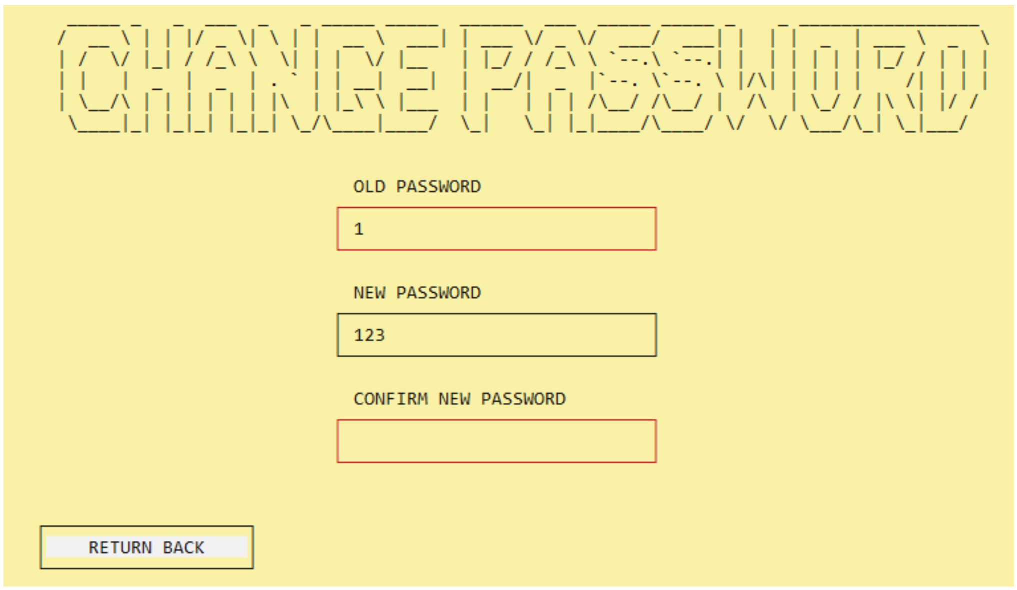


Figure II.2.a.3: We can return back whenever we need

b. Log in/Log out

```
bool isLoggedIn();
void login(bool isStudent, string &username, bool &opt);
void logout();
```

These are the prototypes of helper functions that support login and logout for both staff and student via file **"isLoggedIn.txt".**

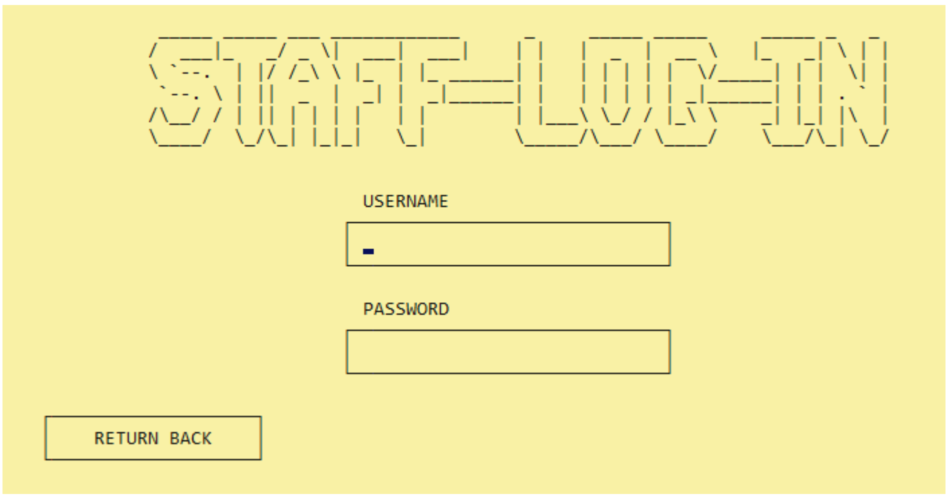


Figure II.2.b.1: Log in Screen

3. Staff

a. Main Screen

void Main_Staff_Screen(string &username, Year* &year_head): This is a function that creates a menu screen for staff members. The screen has options to **"LOG OUT", "VIEW PROFILE", "ADD A NEW SCHOOL YEAR", "ACCESS A SCHOOL YEAR", and "CHANGE PASSWORD".** The program uses console output functions to create a graphical interface for the menu, including boxes and text. The program also uses **keyboard input** to allow the user to select an option and change the highlighted option on the screen. Once an option is selected, the program will perform the corresponding action, such as displaying a profile or creating a new school year.

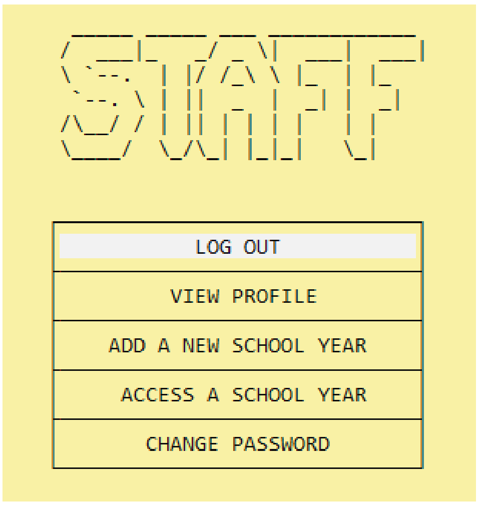
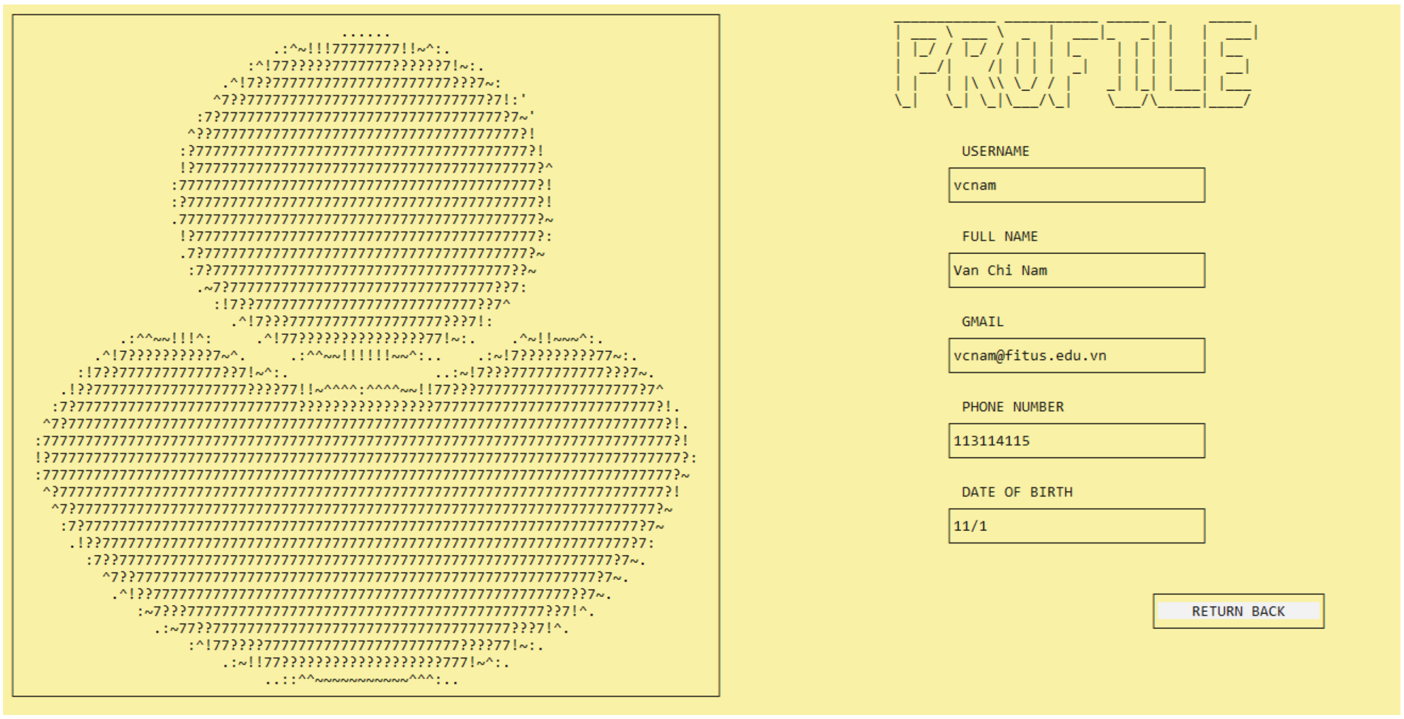


Figure II.3.a.1: Main Staff Screen

void staff_info(string username, Year* &year_head): The function displays all of the staff info by iterating through **staff_info.csv** and match the username before presenting the information in a formatted way.



USERNAME
vcnam

FULL NAME
Van Chi Nam

GMAIL
vcnam@fitus.edu.vn

PHONE NUMBER
113114115

DATE OF BIRTH
11/1

Figure II.3.a.2: Staff's profile

b. Add a new school year

Year* getYearListFromFile(): This is a helper function that reads the **"schoolyear.txt"** file and creates a **linked list of Year** nodes based on the contents of the file. Each Year node represents a school year and contains various information stored in struct Year. The function returns a pointer to the head of the linked list of Year nodes, or nullptr if there was an error reading the file or the file is empty.

i. Checking functions

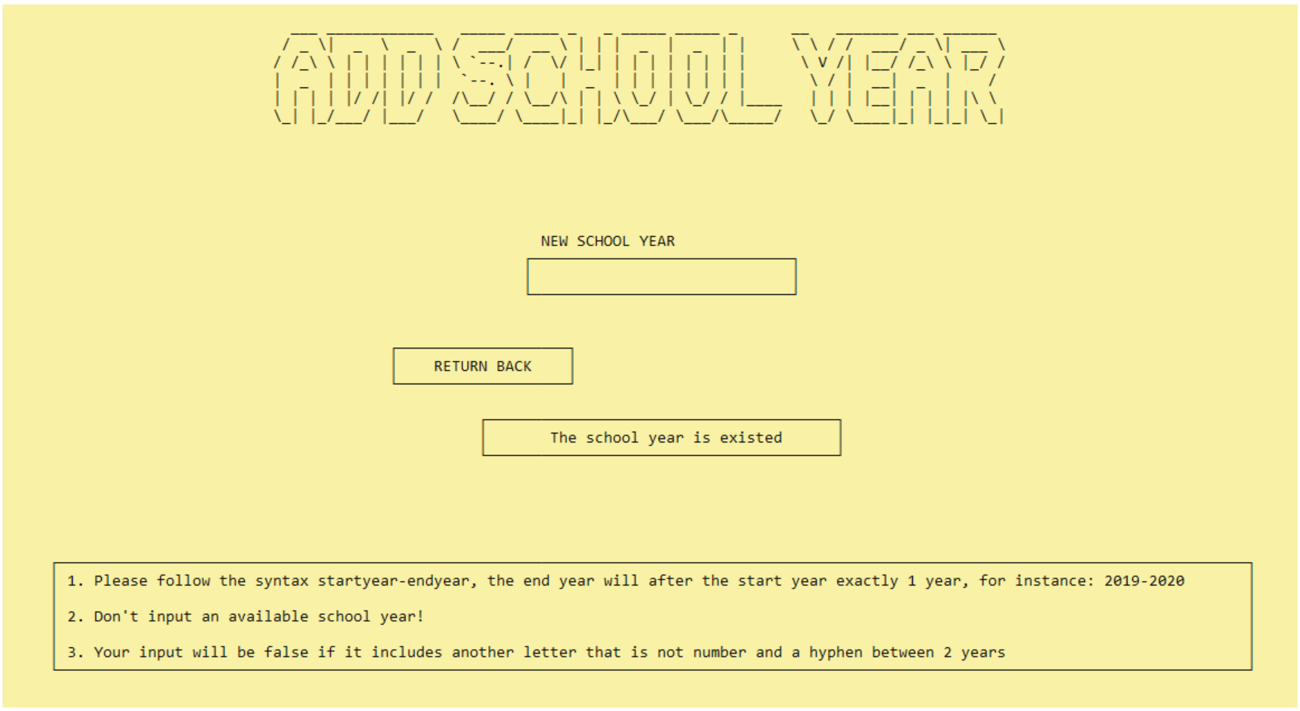
```
bool checkSyntaxOfSchoolYear (string year_name);
bool checkExistingYear (Year* &year_head, string year_name);
```

bool checkSyntaxOfSchoolYear(string year_name): This function takes a string year_name as input and checks if it is in the correct format of **"start_year-end_year"**. If the input string violates any of the three principles **(not having the correct format, having an existing year, or not being a one-year difference),** it returns false. Otherwise, it returns true.

bool checkExistingYear(Year* &year_head, string year_name): This function takes a pointer year_head to the head of the linked list and a string year_name as input and checks if the linked list already contains a year with that name. If it does, it returns false. Otherwise, it returns true.

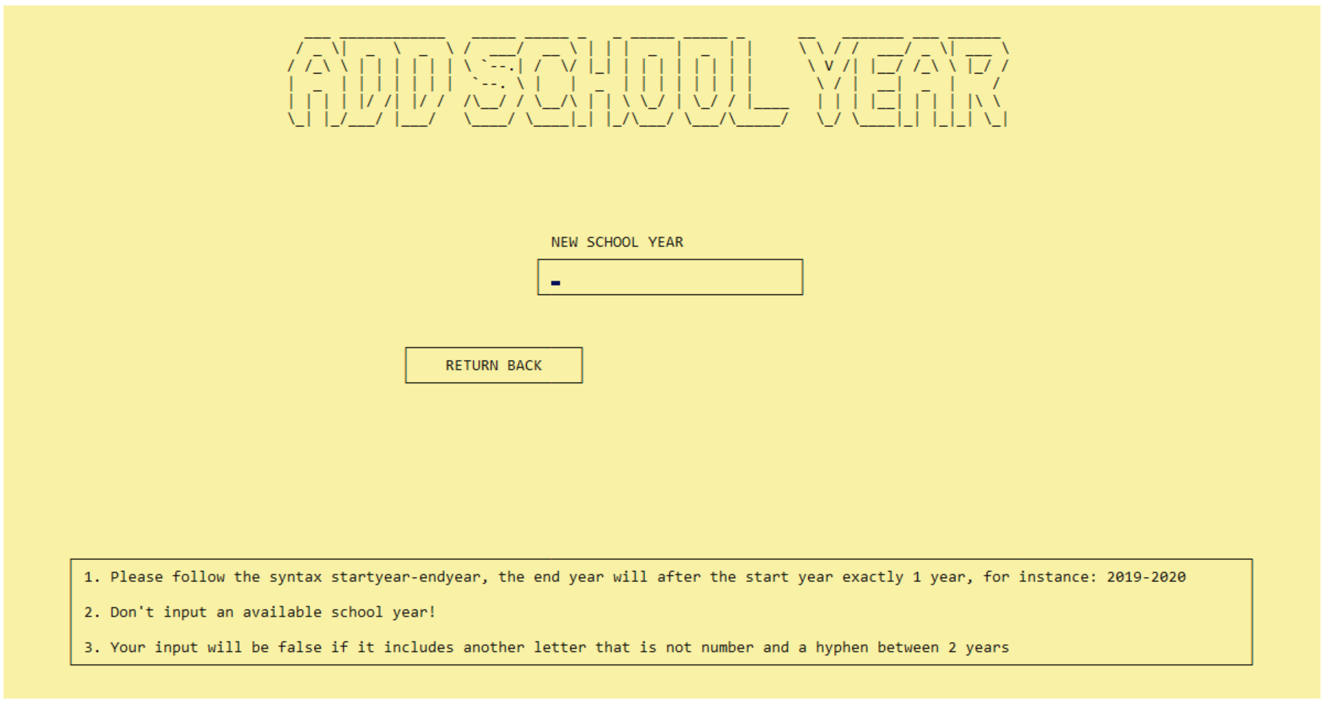
ii. Add a school year

void addNewSchoolYear(Year* &year_head): This function adds a new school year to the linked list of school years. It first creates a new node for the linked list and then prompts the user to input the name of the new school year. It then calls the **checkSyntaxOfSchoolYear(string year_name)** and **checkExistingYear(Year* &year_head, string year_name)** functions to check if the input is valid. If the input is valid, the function adds the new node to the linked list and writes the new year to a file. Otherwise, it displays an error message and prompts the user to input again.



```
The school year is existed

1. Please follow the syntax startyear-endyear, the end year will after the start year exactly 1 year, for instance: 2019-2020
2. Don't input an available school year!
3. Your input will be false if it includes another letter that is not number and a hyphen between 2 years
```

Figure II.3.b.ii.1: AddNewSchoolYear Screen



```
-

1. Please follow the syntax startyear-endyear, the end year will after the start year exactly 1 year, for instance: 2019-2020
2. Don't input an available school year!
3. Your input will be false if it includes another letter that is not number and a hyphen between 2 years
```

Figure II.3.b.ii.2: Error message

c. Access a school year

void viewSchoolYear_Screen(string username, Year* year_head): This function provides a comprehensive list of accessible school years for the given user, while also providing a **"RETURN BACK"** button for easy navigation back to the previous pages.

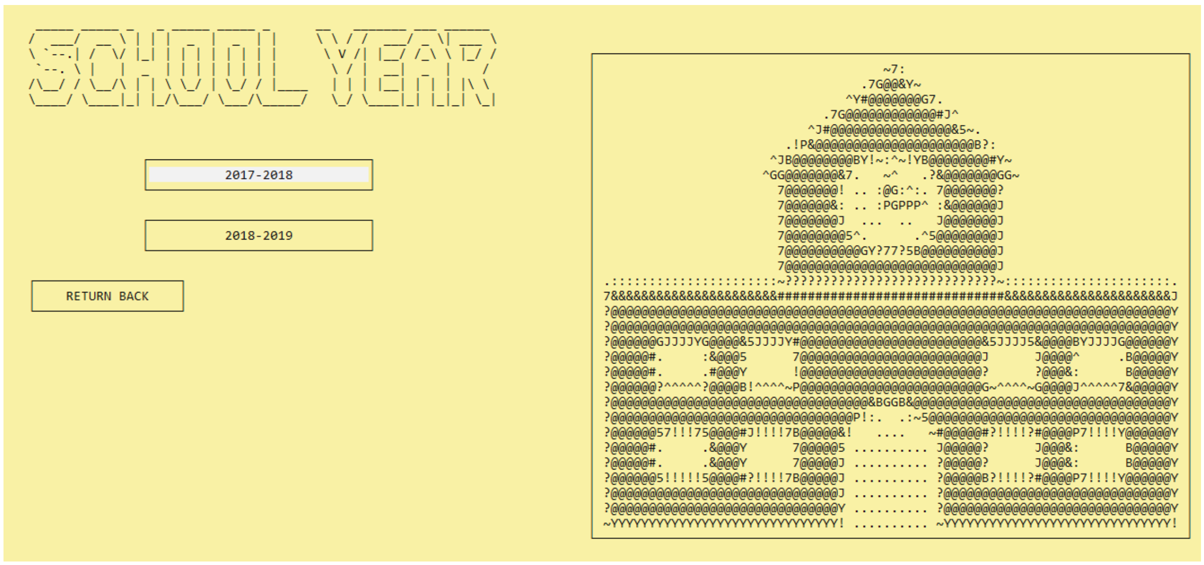


Figure II.3.c.1: The screen of choosing a year

void accessSchoolYear(string username, Year* &year_head): This function presents the user with **two distinct menus** that allow a staff member to interact with semesters and classes, offering a range of actions for each category.

Semester functions include:	Class functions include:
ADD SEMESTER	**ADD CLASS**
ACCESS SEMESTER	**ACCESS CLASS**
RETURN BACK	**RETURN BACK**

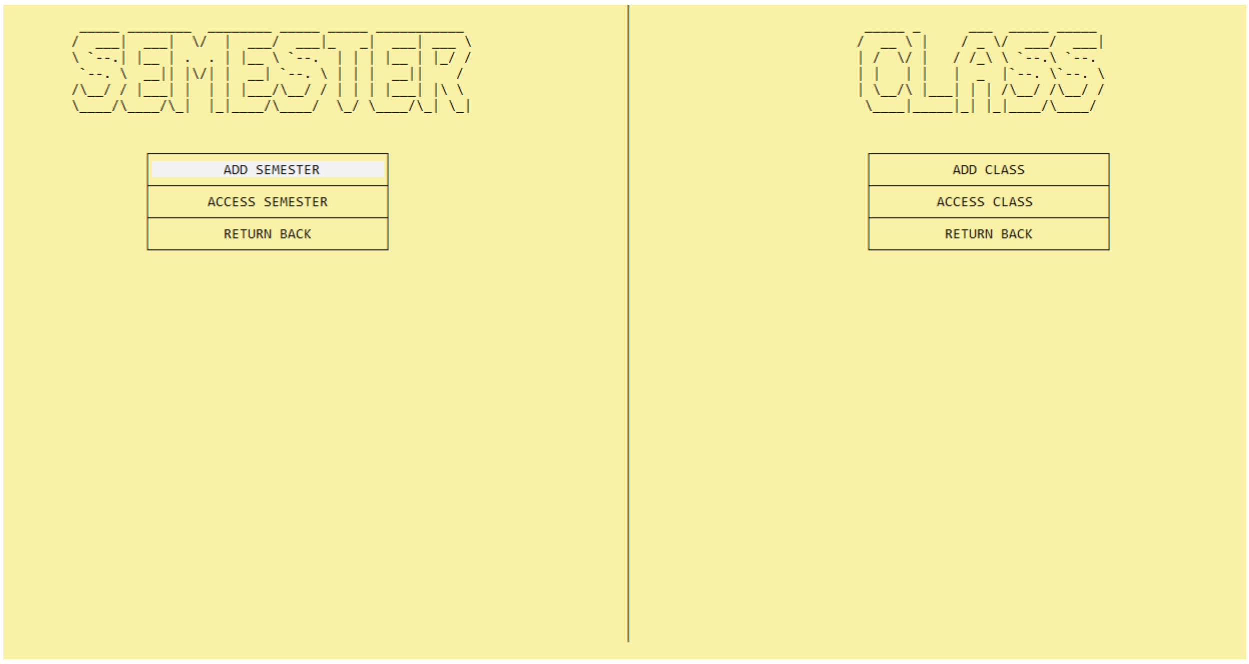


Figure II.3.c.2: Access a School Year

d. Add a new semester

Semester* getSemesterListFromFile(Year* &year_head): This is a helper function that reads a file named **string file_name = year_head->year_name + "_semester.txt"** where year_name is the name of **a year obtained from a linked list of Year nodes** passed as a parameter to the function. The function then creates a linked list of Semester nodes based on the contents of the file. Each Semester node represents a semester and contains various information stored in struct Semester. The function returns a pointer to the head of the linked list of Semester nodes for the given year, or nullptr if there was an error reading the file or the file is empty.

i. Checking functions

```
bool checkExistingSemester (Semester* &semester_head, int semester_ord);
bool checkInvalidDayOfDate (int year, int month, int day);
bool checkStartDateAndEndDate (int startyear, int startmonth, int startday, int
endyear, int endmonth, int endday);
```

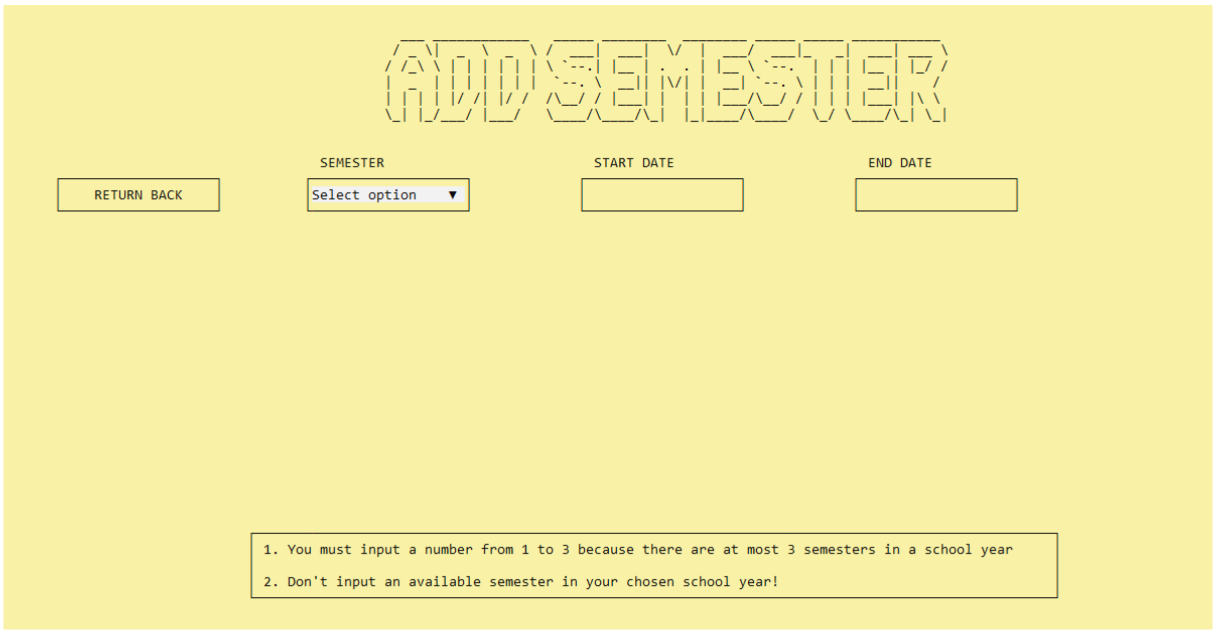
bool checkExistingSemester (Semester* &semester_head, int semester_ord): This function checks if a semester with a given semester ordinal already exists in the linked list of semesters. If such a semester exists, it returns false; otherwise, it returns true.

bool checkInvalidDayOfDate (int year, int month, int day): This function checks if a given day is valid for a given year and month, taking into account whether or not the year is a leap year. If the day is invalid, the function returns false; otherwise, it returns true.

bool checkStartDateAndEndDate (int startyear, int startmonth, int startday, int endyear, int endmonth, int endday): This function checks if a given start date and end date are valid. Specifically, it checks if the start date is before the end date and if the years and months of the start and end dates are valid. If the start and end dates are valid, the function returns true; otherwise, it returns false.

ii. Add a new Semester

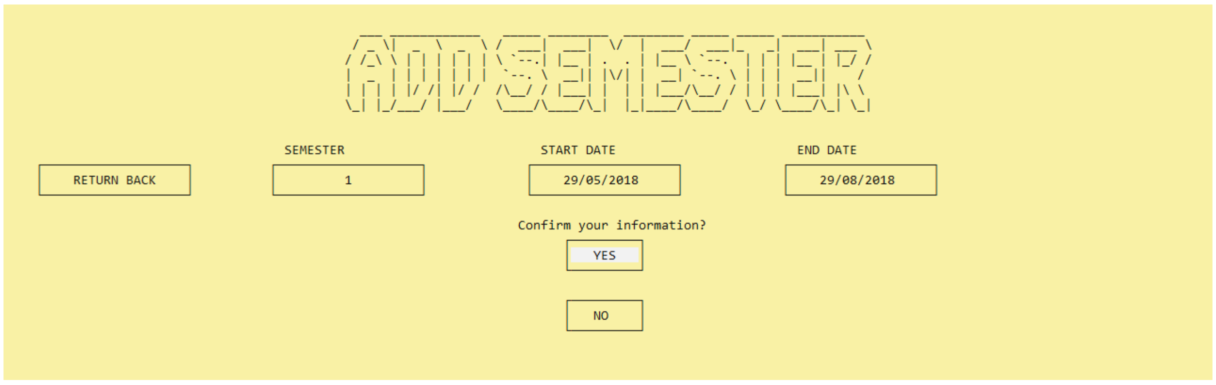
void addNewSemester (string username, Year* &year_head): This function prompts the user to input the semester, start date, and end date for a new semester, and then stores these values. It uses the **checkExistingSemester, checkInvalidDayOfDate, and checkStartDateAndEndDate** helper functions to validate the user input. This function stores all these values in **string file_name = year_head->year_name + "_semester.txt"** and create a CSV file for future storage of courses **(string name_file = year_head->year_name + "_semester" + (char)(temp_semester1->Semester_Ord + 48) + "_course.csv").**



SEMESTER START DATE END DATE

1. You must input a number from 1 to 3 because there are at most 3 semesters in a school year
2. Don't input an available semester in your chosen school year!

Figure II.3.d.ii.1: AddNewSemester Screen



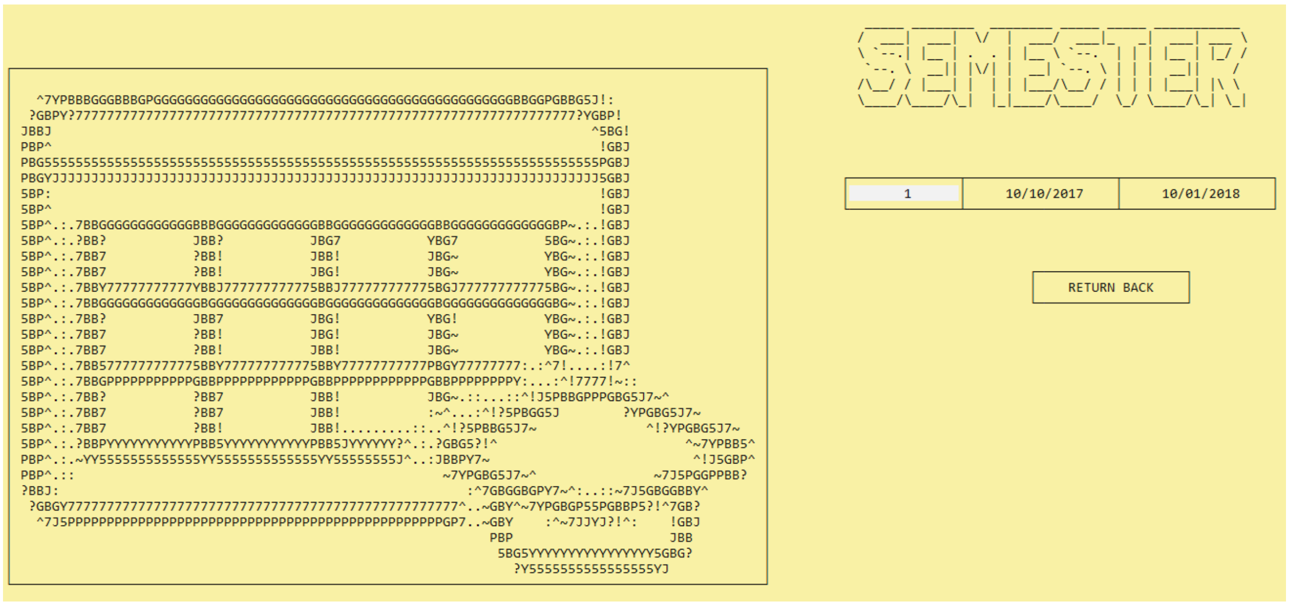
SEMESTER	START DATE	END DATE
1	29/05/2018	29/08/2018

Confirm your information?

Figure II.3.d.ii.2: Comfirm information one more time

e. Access a semester

void viewSemester_Screen(string username, Year* year_head): This function provides a comprehensive list of accessible terms for the given user, while also providing a **"RETURN BACK"** button for easy navigation back to the previous pages.



1	10/10/2017	10/01/2018

Figure II.3.e.1: The screen of choosing a semester

void accessSemester(string username, Year* &year_head, Semester* &semester_head): This function generates a user interface that allows for the management of courses in a designated semester. The interface presents a set of options, including **"ADD A COURSE", "ACCESS A COURSE", "UPDATE A COURSE" and "RETURN BACK"** with the help of **getSemesterListFromFile helper** function.

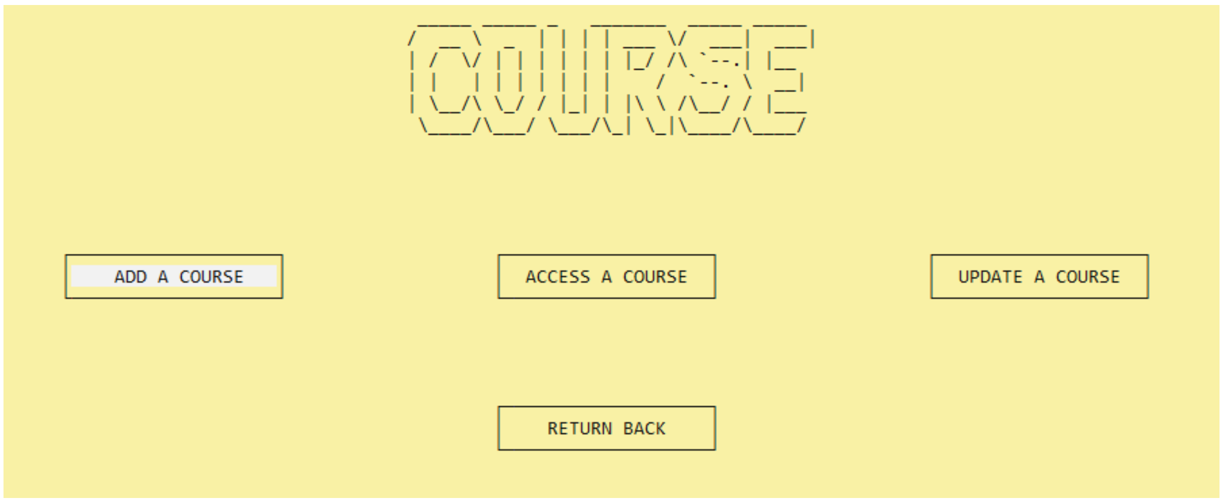


Figure II.3.e.1: Access a Semester

f. Add a new course

Course* getCourseListFromFile(Year* year_head, Semester* &semester_head): This function reads a CSV file containing course information for a given year and semester (***string file_name = year_head->year_name + "_semester" + (char)(semester_head->Semester_Ord + 48) + "_course.csv")***. It creates a linked list of courses by parsing the file and extracting the relevant details for each course. The function returns a pointer to the head of the linked list of courses.

i. Helper functions

```
string NormalizeCourseID (string Course_ID);
string Normalization (string input);
string NormalizeClassName (string Class_Name);
```

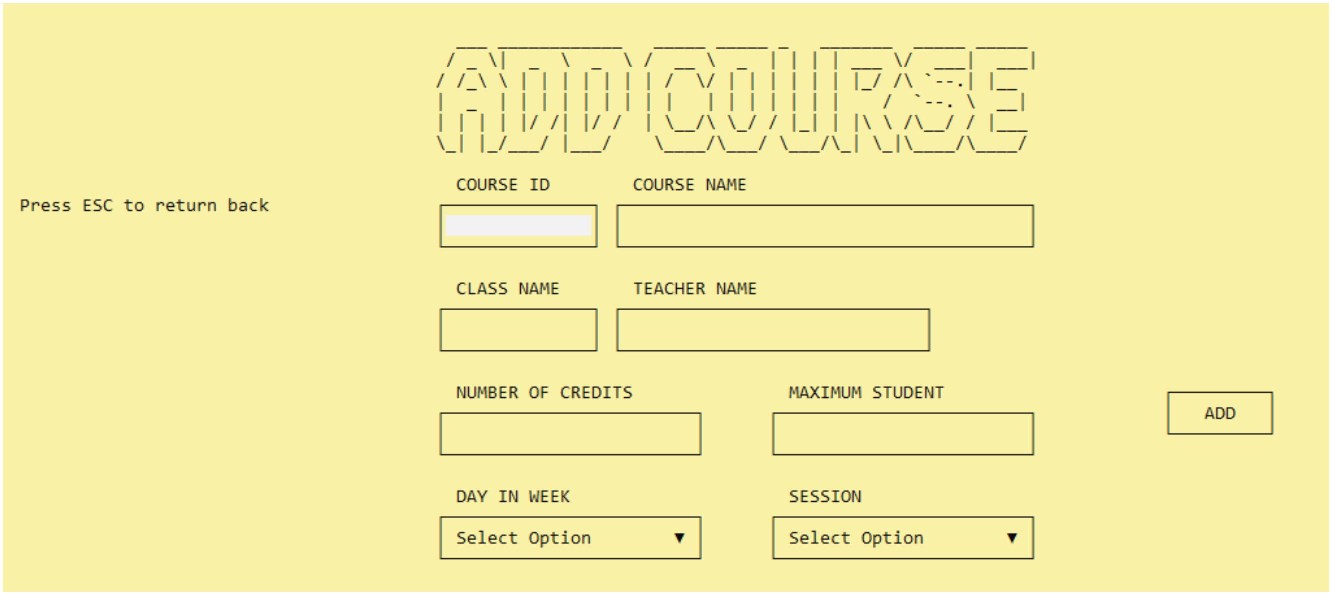
string NormalizeCourseID (string Course_ID): This function converts any lowercase letters in a given course ID string to uppercase letters, returning the normalized course ID as a string. It does this by iterating through each character in the string and converting any lowercase letters to uppercase letters.

string Normalization (string input): This function normalizes a string by converting it to title case and removing any leading or trailing whitespace. It checks for and removes leading whitespace and iterates through each character in the input string, converting the first character and any following characters after a space character to uppercase and the rest to lowercase. It also removes any excess whitespace between words. Finally, it returns the normalized string.

string NormalizeClassName (string Class_Name): This function normalizes a string by converting all lowercase characters to uppercase. It iterates through each character in the input string, checking if it is a lowercase character, and if so, converts it to uppercase. Finally, it returns the normalized string.

ii. Add a new course

void addNewCourse (string username, Year* year_head, Semester* semester_head): This function provides a streamlined user interface for creating a new course, with clear prompts and input requirements. Then, it updates ***string file_name = year_head->year_name + "_semester" + (char)(semester_head->Semester_Ord + 48) + "_course.csv"***; ***string filename = Class_name + "_" + "Semester" + char_semester + "_" + year_head->year_name + "_courses.csv"*** while also creating ***string name_file = temp_course1->course_ID + "_Semester" + (char)(semester_head->Semester_Ord + 48) + "_" + ClassName + "_" + year_head->year_name + "_Scoreboard.csv"*** and ***string name_file = temp_course1->course_ID + "_Semester" + (char)(semester_head->Semester_Ord + 48) + "_" + ClassName + "_" + year_head->year_name + "_ student.csv"*** for future usage.


Press ESC to return back

COURSE ID	COURSE NAME
CLASS NAME	TEACHER NAME
NUMBER OF CREDITS	MAXIMUM STUDENT
DAY IN WEEK: Select Option	SESSION: Select Option

Figure II.3.f.ii.1: AddCourse Screen

g. Update a course

i. Access the screen of updating Course

void viewUpdateCourseInformation (string username, Year* year_head, Semester* semester_head): This function provides a comprehensive list of accessible courses for the given user, while also providing a **"RETURN BACK"** button for easy navigation back to the previous pages

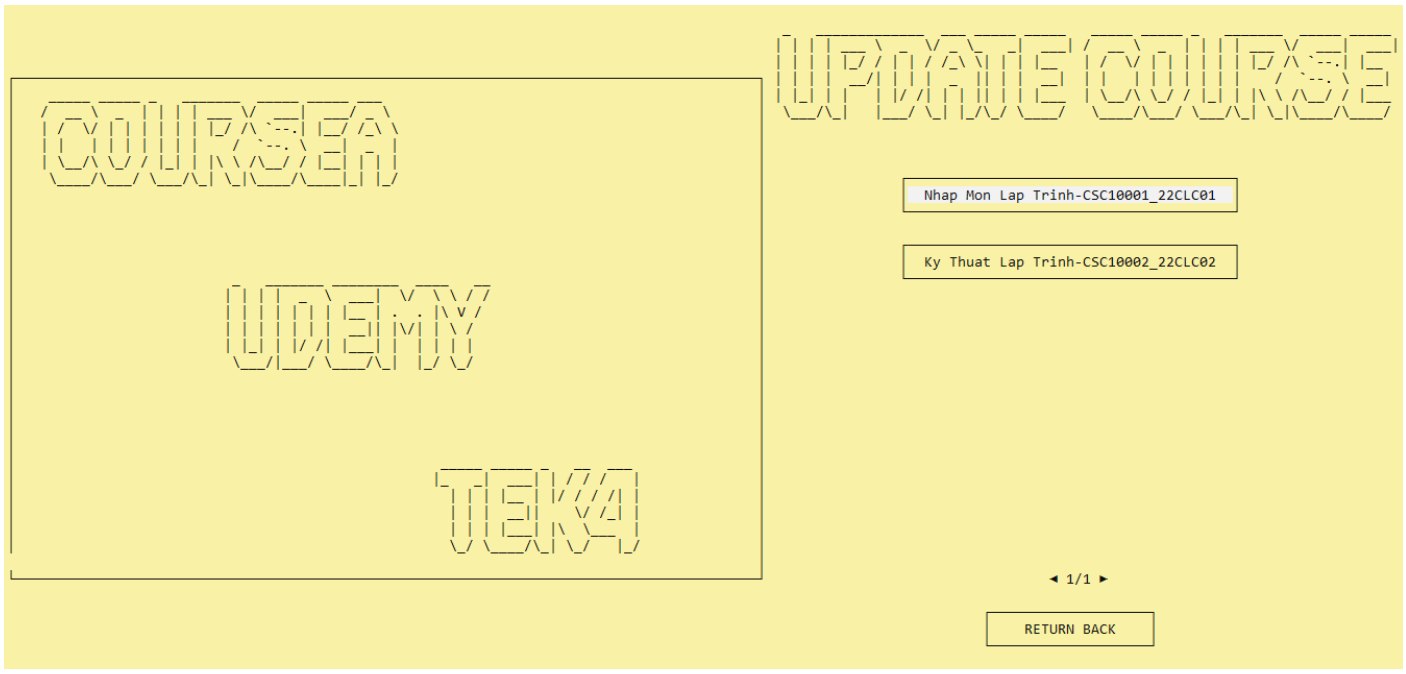

COURSERA
UDEMY
TEK4
UPDATE COURSE
Nhap Mon Lap Trinh-CSC10001_22CLC01
Ky Thuat Lap Trinh-CSC10002_22CLC02
◂ 1/1 ▸
RETURN BACK

Figure II.3.g.i.1: Choose a course to update

void updateACourse (string username, Year* year_head, Semester* semester_head, Course* course_head): The function presents the user with a comprehensive menu of options to facilitate their management of a particular course. Upon selecting a desired option and pressing the "ENTER" key, the corresponding function associated with that option is called and executed to effect the necessary changes.

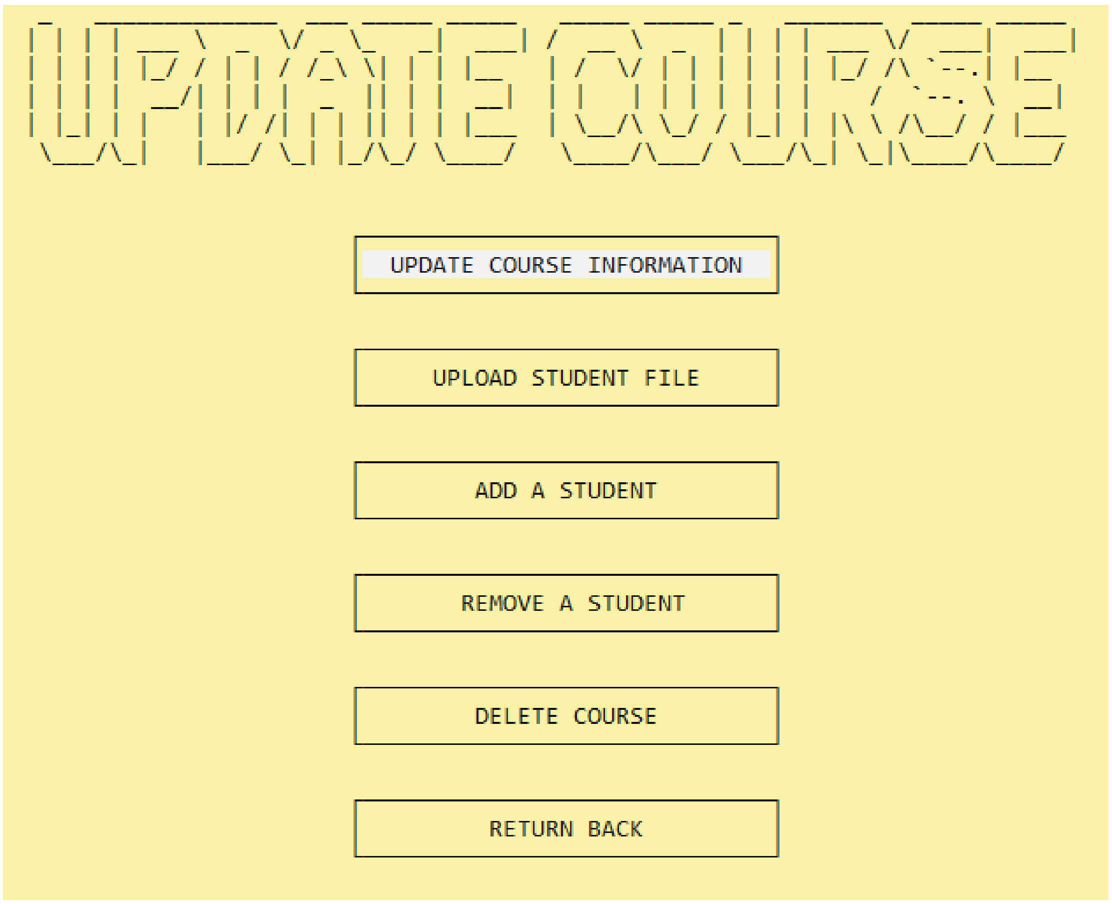
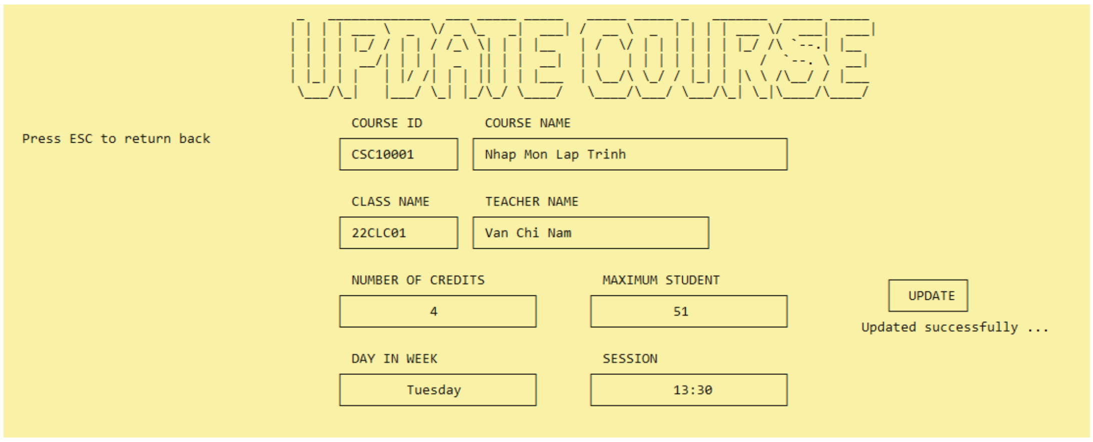


Figure II.3.g.i.2: The list of course updating options

ii. Update Course's information

void updateInformationOfCourse (string username, Year* year_head, Semester* semester_head, Course* accessed_course): the function enables the user to simultaneously modify a range of diverse details associated with a given course, such as the course ID, the class name, the teacher name, the course name, the number of credits, the maximum number of students, the day of the week, and the session. After the user has made the necessary adjustments, the updated information is subsequently saved to related files (***string file_name = year_head->year_name + "_semester" + (char)(semester_head->Semester_Ord + 48) + "_course.csv" + string file_name1 = temp->class_name + "_semester" + (char)(semester_head->Semester_Ord + 48) + "_" + year_head->year_name + "_courses.csv")***.


UPDATE COURSE

Press ESC to return back

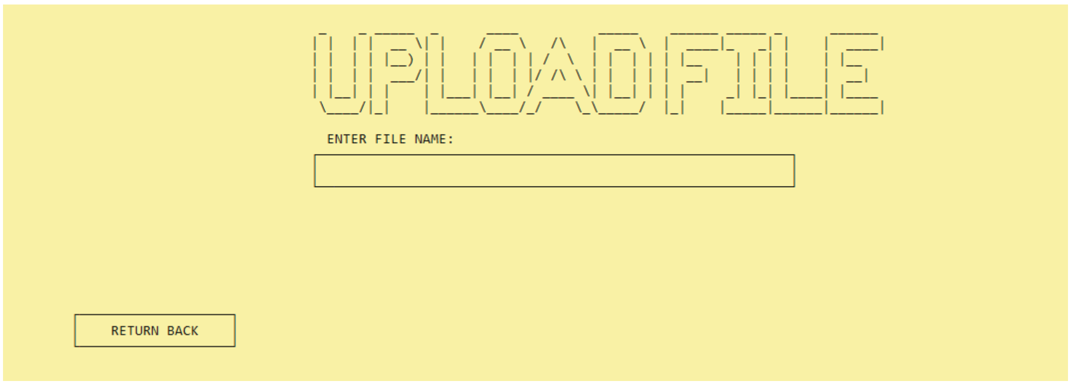
COURSE ID: CSC10001
COURSE NAME: Nhap Mon Lap Trinh
CLASS NAME: 22CLC01
TEACHER NAME: Van Chi Nam
NUMBER OF CREDITS: 4
MAXIMUM STUDENT: 51
DAY IN WEEK: Tuesday
SESSION: 13:30

Updated successfully ...

Figure II.3.g.ii.1: Update information successfully

iii. Upload Student File

void uploadStudentFileToCourse (string username, Year* year_head, Semester* semester_head, Course* course_head): This function allows teacher to type in their file name that contains information of students enrolled in the course, all of which will then be appended to ***string name_file = course_head->course_ID + "_Semester" + (char)(semester_head->Semester_Ord + 48) + "_" course_head->class_name + "_" + year_head->year_name + ".csv"***.


UPLOAD FILE

ENTER FILE NAME:

Figure II.3.g.iii.1: Enter file name for uploading students to Course

iv. Add a Student

Helper Functions

```
int numOfStudent(Course* course);
bool isStudentInCourse(Course* course, Student* student);
Student* findStudentInClass(Class* class_head, string studentID);
void addCourseToStudent(Student* &student, Course* course);
```

int numOfStudent(Course* course): This function **counts the number of students** enrolled in a Course by iterating through the linked list of students and incrementing a counter variable. The function returns the final value of the counter as the total number of students in the course.

bool isStudentInCourse(Course* course, Student* student): This function checks if a given Student is enrolled in a specified Course by iterating through the linked list of students in the course and **comparing the student_ID** of each student to that of the given student. If there is a match, the function returns true; otherwise, it returns false.

Student* findStudentInClass(Class* class_head, string studentID): The function iterates through a linked list of Class objects to find a student with a given student_ID. If a match is found, the function creates a new Student object and sets its properties with the corresponding values from the matched student in the linked list. Finally, the function returns a pointer to the new Student object. If no match is found, the function returns a nullptr. The function can be used to search for a specific student in a linked list of classes and retrieve their properties.

void addCourseToStudent(Student* &student, Course* course): This function adds a new course to a student's list of courses. If the student has no courses, the function sets the course_head pointer to the new course. Otherwise, it adds the new course to the end of the list.

Add a student

void addStudentToCourse(string username, Course* &course, Year* &year_head, Semester* semester_head): This function is designed to allow users to add a student to a course. It takes input from the user for the student ID, and with help of the other support functions mentioned above, checks if the course has reached the maximum number of students allowed, and adds the student to the course if there is still room. The function also saves the changes to **string file_name = course->course_ID + "_Semester" + (char)(semester_head->Semester_Ord + 48) + "_" + course->class_name + "_" + year_head->year_name + "_student.csv".**

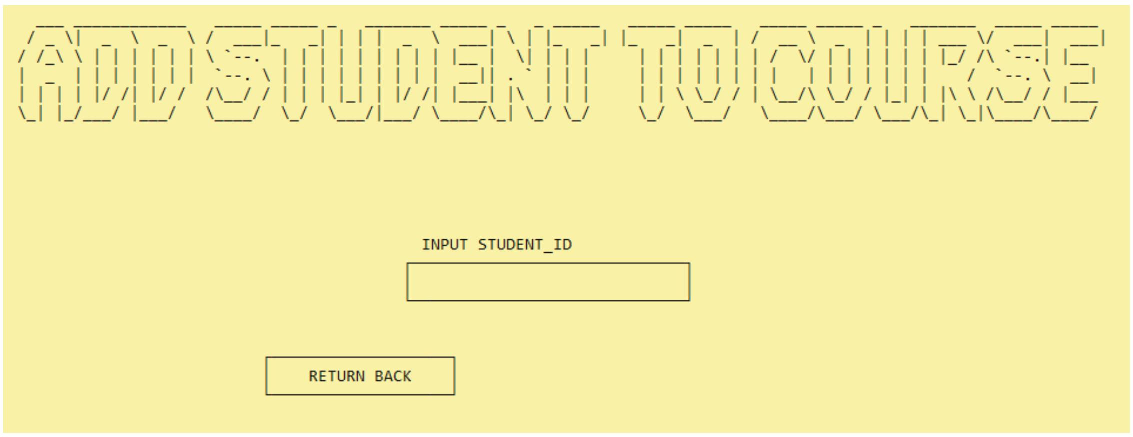


Figure II.3.g.iv.1: Input Student ID of student needed adding

v. Remove a Student

Helper Functions

```
string printDate(Date a);
void deleteStudent(Student* &student_head, string studentID);
Student* getListStuFromFile(string filename);
```

string printDate(Date a): This function takes a Date object and returns a string in the format "dd/mm/yyyy" representing the date.

void deleteStudent(Student *&student_head, string studentID): This function takes a student ID and a pointer to the head of a linked list of Student objects, and removes the student with that ID from the list if it exists.

Student* getListStuFromFile(string filename): This function takes a filename, reads a list of students from a CSV file with that name, and returns a pointer to the head of a linked list of Student objects.

Remove Student

void removeStudent(string filename, string studentID): This function takes a filename and a student ID, reads a list of students from the file, removes the student with the given ID, and writes the updated list to a temporary file. The original file is then deleted, and the temporary file is renamed to the original filename.

void removeStudent(string filename, string studentID): This function takes a filename and a student ID, reads a list of students from the file, removes the student with the given ID, and writes the updated list to a temporary file. The original file is then deleted, and the temporary file is renamed to the original filename.

void removeStudentFromCourse(string username, Course* &course, Year* &year_head, Semester* semester_head): This function prompts the user to input a student ID to remove from the course's list of students. If the student is not found in the course, an error message is printed to the console. The function also removes the student from the file associated with the course.

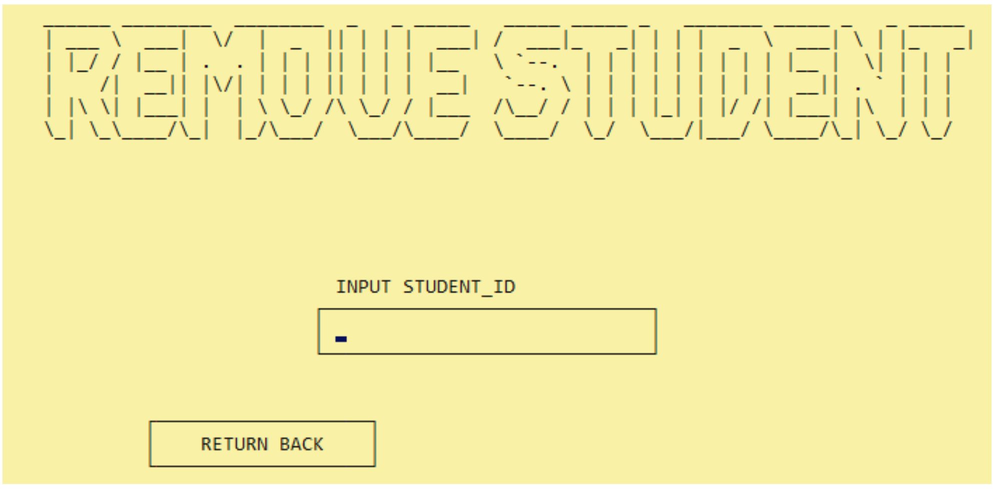


Figure II.3.g.v.1: Input Student ID of student needed removing

vi. Delete Course

<u>Helper Functions</u>

```
void addTailCourse(Course* &course_head, Course* tmp);
void removeCourseFromList(Course* &course_head, Course* tmp);
Course* getCourseFromFile(string filename);
void deleteCoursefromFile(string filename, Course* course_head);
```

void addTailCourse(Course* &course_head, Course* tmp): The function adds the course object to the end of the linked list.

void removeCourseFromList (Course* &course_head, Course* tmp): The function finds **the course tmp** in the linked list of **course_head** and removes it if **tmp** was found

Course* getCourseFromFile(string filename): The function searches the linked list for a course object with the given course ID and class name, removes that object from the list, and deallocates the memory used by the object. If no object with the given course ID and class name is found in the list, the function does nothing.

void deleteCoursefromFile(string filename, string courseid, string nameclass): This function deletes a course with a given course ID and class name from a file. It reads the course information from the file and stores it in a linked list. Then, it removes the course from the linked list. After that, it writes the remaining courses in the linked list to a temporary file. Finally, it deletes the original file and renames the temporary file to the original filename.

<u>Delete Course</u>

void deleteCourse(string username, Semester *semester_head, Year* year_head): This function prompts the user to confirm their intention to delete a course by asking for the course ID and class name. It then retrieves the data from a file, removes the specified course from the list, and writes the updated list to a temporary file. Finally, it deletes the original file and renames the temporary file to the original filename.

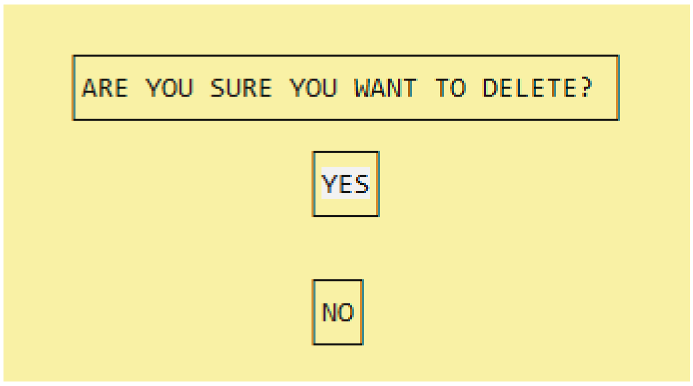


Figure II.3.g.v.1: Comfirm whether users will delete course

h. Access a course

void viewCourse_Screen(string username, Year* year_head, Semester* semester_head): Similar with the screen of **viewUpdateCourse**, it also shows the list of accessible courses for users to choose.

int COURSE_Interface (string username, Year* &year_head, Semester* &semester_head, Course* &course_head): This function displays actions that can be taken to manage a course with options including **"VIEW ALL STUDENTS IN THIS COURSE", "UPDATE SCOREBOARD FOR THIS COURSE", "EXPORT SCOREBOARD.CSV FOR TEACHER", "IMPORT SCOREBOARD.CSV TO THIS SYSTEM" and "RETURN BACK"** which returns a integer value that represents the chosen course of action.

void accessCourse(string username, Year* &year_head, Semester* &semester_head, Course* &course_head): This function employs the use of COURSE_Interface as a means to obtain an integer value, which is subsequently utilized to select and execute the appropriate function.

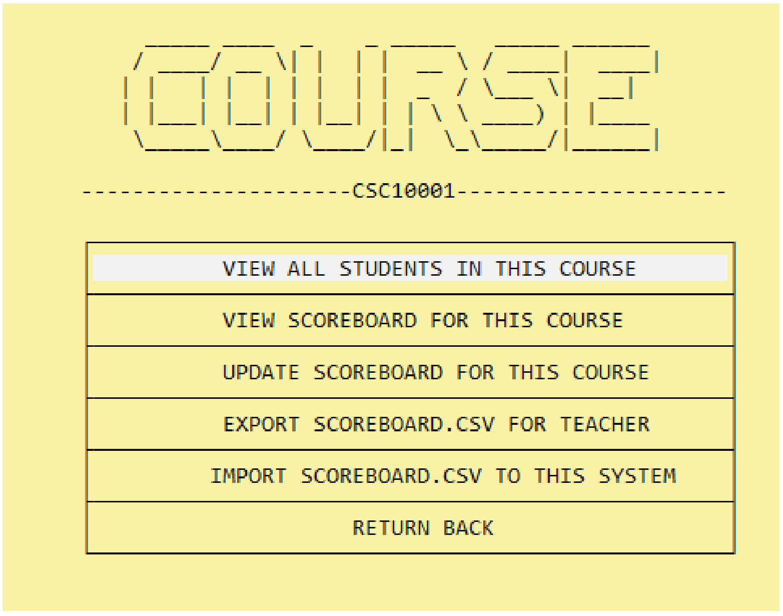
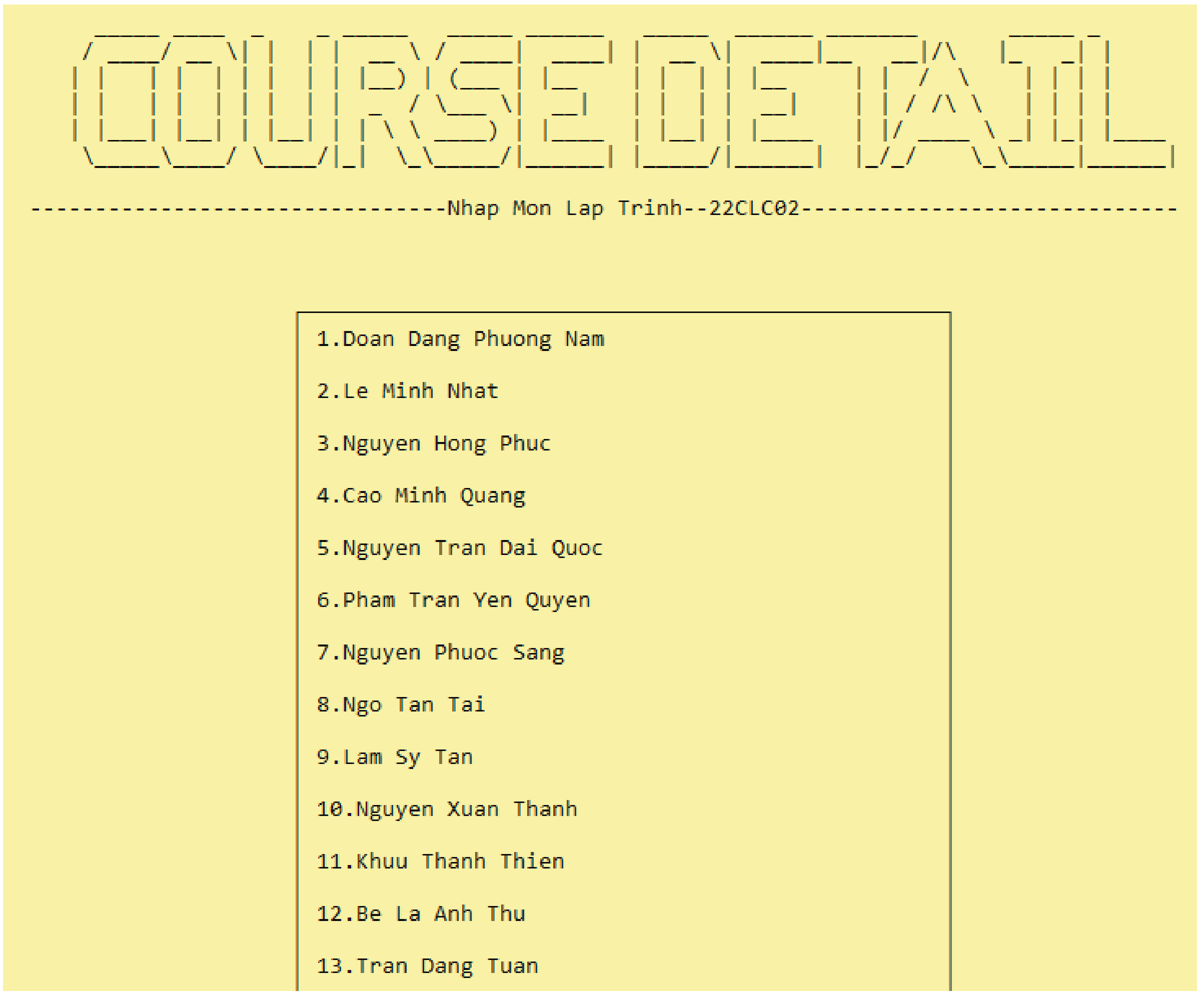

COURSE
--------------------CSC10001--------------------
VIEW ALL STUDENTS IN THIS COURSE
VIEW SCOREBOARD FOR THIS COURSE
UPDATE SCOREBOARD FOR THIS COURSE
EXPORT SCOREBOARD.CSV FOR TEACHER
IMPORT SCOREBOARD.CSV TO THIS SYSTEM
RETURN BACK

Figure II.3.h.1: The screen of accessing a course

i. View Students in Course

void viewStudentInCourse(string username, Year* &year_head, Semester* &semester_head, Course* &course_head): This function displays a header and **a list of students enrolled in a specific course**, which is passed to the function as a parameter. If there are no students in the course, a message is displayed. If there are students in the course, their names are displayed in a box. The function also provides an option for the user to return to the previous menu by pressing Enter.



```
-------------------------------Nhap Mon Lap Trinh--22CLC02----------------------------

 1.Doan Dang Phuong Nam
 2.Le Minh Nhat
 3.Nguyen Hong Phuc
 4.Cao Minh Quang
 5.Nguyen Tran Dai Quoc
 6.Pham Tran Yen Quyen
 7.Nguyen Phuoc Sang
 8.Ngo Tan Tai
 9.Lam Sy Tan
 10.Nguyen Xuan Thanh
 11.Khuu Thanh Thien
 12.Be La Anh Thu
 13.Tran Dang Tuan
```

Figure II.3.h.i.1: The list of students in a course

ii. View Course's Scoreboard

<u>Helper functions</u>

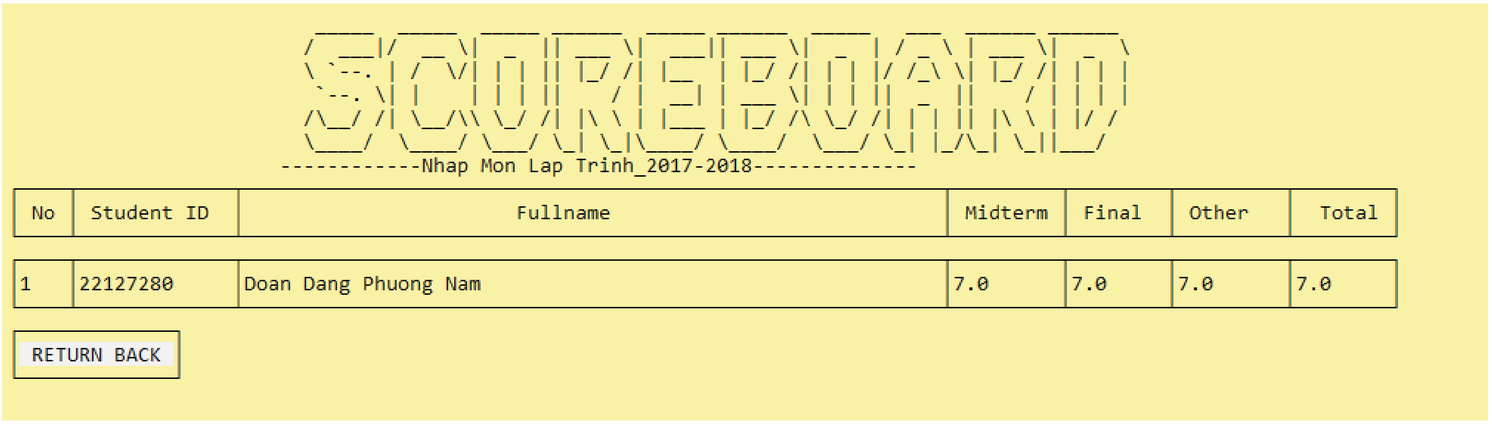
```
string format_float(float value);
void printStudentInfo(int y, int counter, string student_id, string fullname,
float midterm, float final1, float other, float total);
```

string format_float(float value): This function takes a floating-point value as input and returns a string representation of that value with a single decimal point.

void printStudentInfo(int y, int counter, string student_id, string fullname, float midterm, float final1, float other, float total): The function is used to print the information of a student in a formatted way on the console.

View Course's Scoreboard

void viewScoreBoard_Course(string filename, string username, Year* &year_head, Semester* &semester_head, Course* &course_head): This function utilizes the helper functions to print out all of the students information stored in string **filename = course_head->course_ID + "_" + "Semester" + ch_semester + "_" + year_head->year_name + "_Scoreboard.csv".**



```
------------Nhap Mon Lap Trinh_2017-2018-------------
```

No	Student ID	Fullname	Midterm	Final	Other	Total
1	22127280	Doan Dang Phuong Nam	7.0	7.0	7.0	7.0

Figure II.3.h.ii.1: A Course's Scoreboard

iii. Export Course's Scoreboard

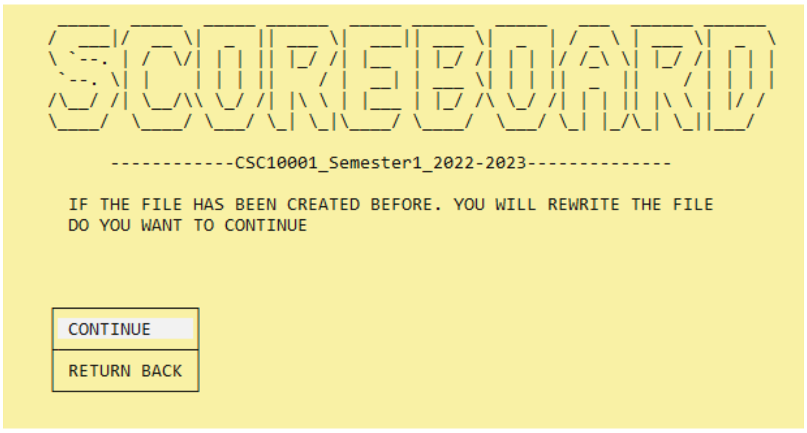
Helper functions

```
Student* getStudentCourseFromFile(Year* &year_head, Semester* &semester_head,
Course* &course_head);
```

Student *getStudentCourseFromFile(Year* &year_head, Semester*&semester_head, Course* &course_head): This function reads student information from a CSV file for a given course, semester, and year. It then creates a linked list of Student objects, adding each student read from the file to the end of the list. Finally, it returns the head of the linked list of students.

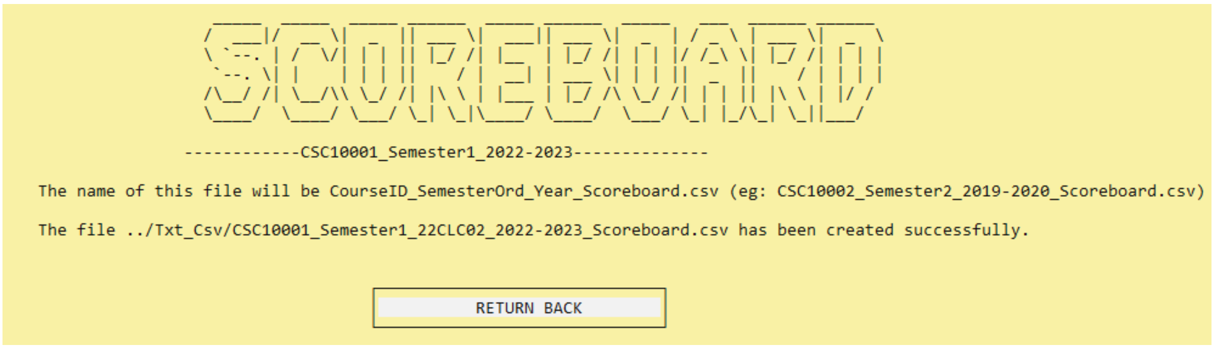
Export Course's Scoreboard

void export_scoreboard(string filename, string username, Year* &year_head, Semester* &semester_head, Course* &course_head): The function exports a course scoreboard to a file, with a given filename. It displays a header with the course name, semester, and year, then checks if there are any students in the course. If there are no students, it displays a message and returns to the access course screen. If there are students, it asks if the staff member wants to continue with the export, and if yes, it writes the scoreboard to a CSV file with the given filename (**string filename = course_head->course_ID + "_" + "Semester" + ch_semester + "_" + course_head->class_name + "_" + year_head->year_name + "_Scoreboard.csv"**). It overwrites the file if it already exists. The function also displays a message when the export is complete.



```
------------CSC10001_Semester1_2022-2023-------------

IF THE FILE HAS BEEN CREATED BEFORE. YOU WILL REWRITE THE FILE
DO YOU WANT TO CONTINUE
```

Figure II.3.h.iii.1: Confirm whether users want to export Scoreboard file



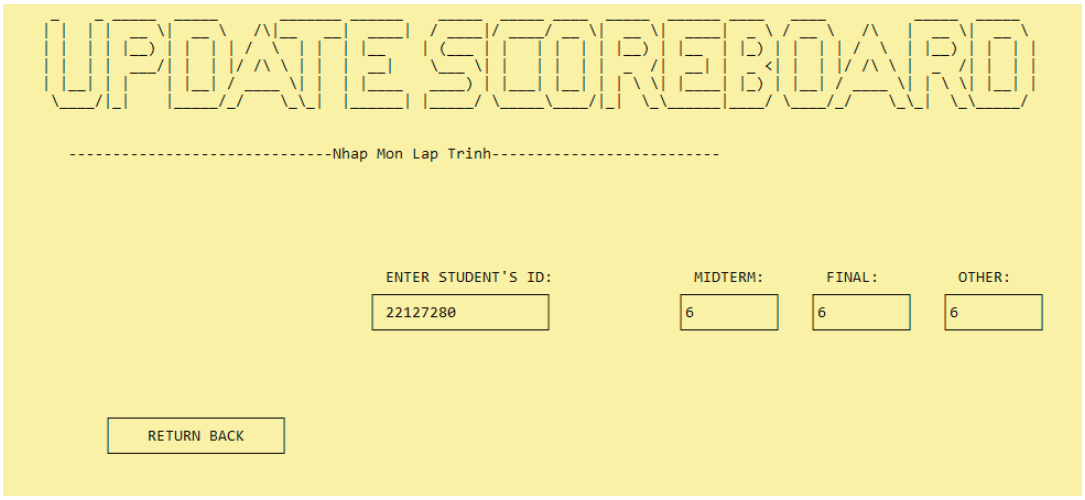
```
------------CSC10001_Semester1_2022-2023-------------

The name of this file will be CourseID_SemesterOrd_Year_Scoreboard.csv (eg: CSC10002_Semester2_2019-2020_Scoreboard.csv)

The file ../Txt_Csv/CSC10001_Semester1_22CLC02_2022-2023_Scoreboard.csv has been created successfully.
```

Figure II.3.h.iii.2: Scoreboard file was exported

iv. Update Course's Scoreboard

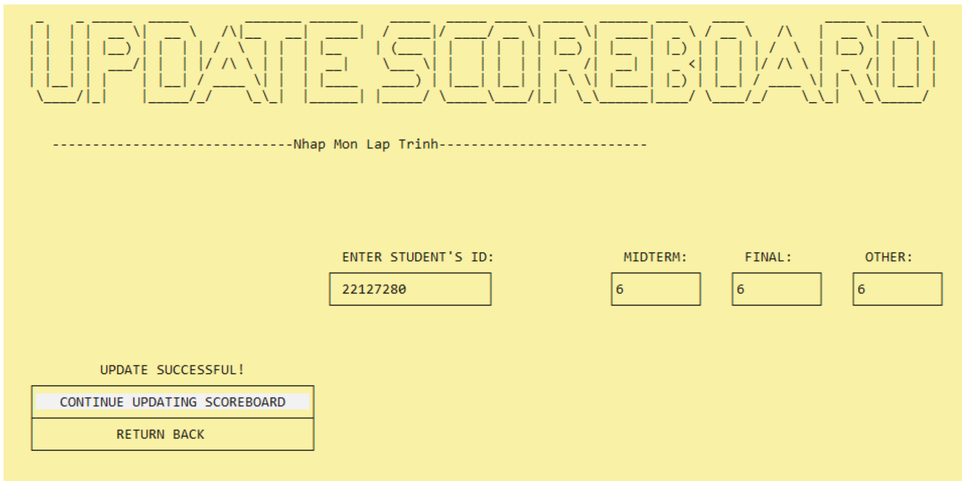
void updateStudentsScore(string filename, string username, Year* &year_head, Semester* &semester_head, Course* &course_head): This function asks user for a student's ID and allow them to change scores while also updating it in a file (**filename = course_head->course_ID + "_" + "Semester" + ch_semester + "_" + course_head->class_name + "_" + year_head->year_name + "_Scoreboard.csv"**).

*Note: The student's ID is validated via **bool check_is_it(string filename, string student_id)** which checks if a student with the given student_id is present in a file specified by filename by searching for the ID in each line of the file.



```
----------------------------Nhap Mon Lap Trinh------------------------

ENTER STUDENT'S ID:      MIDTERM:   FINAL:   OTHER:
22127280                 6          6        6
```

Figure II.3.h.iv.1: Update the scores of a student in a course

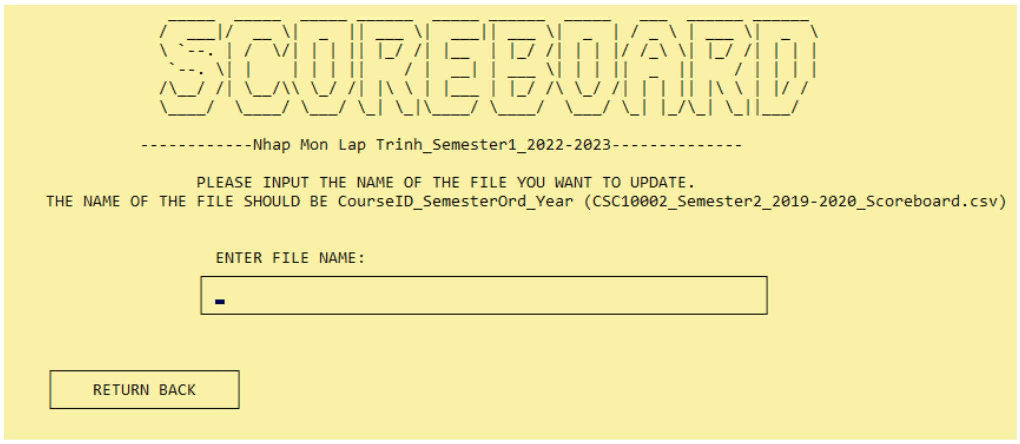


```
-----------------------------Nhap Mon Lap Trinh-------------------------

ENTER STUDENT'S ID:        MIDTERM:    FINAL:    OTHER:
22127280                   6           6         6

UPDATE SUCCESSFUL!
```

Figure II.3.h.iv.2: Updated successfully

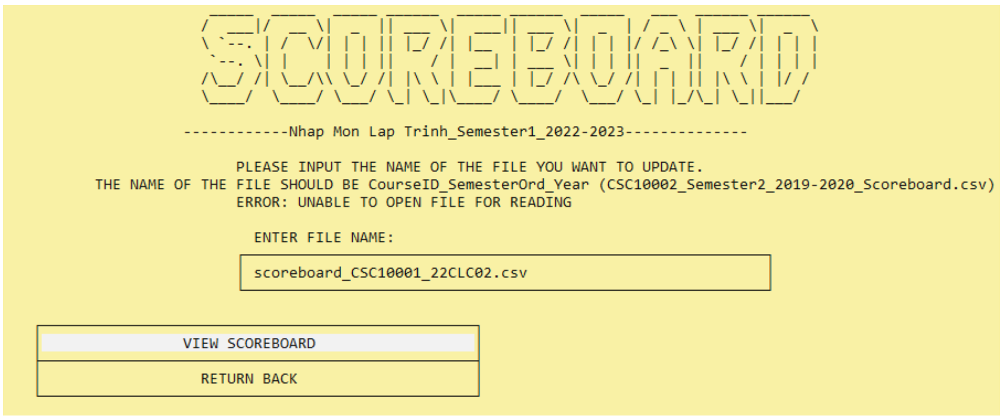
v. Import Course’s Scoreboard

void update_scoreBoard(string filename, string username, Year* &year_head, Semester* &semester_head, Course* &course_head): This function enables teachers to update the scores of students by providing a filename for the corresponding score file. The function checks if the file exists before allowing the teacher to view it on their console or return to previous screen with an additional function called **int choose_interface(string filename, string username, Year* &year_head, Semester* &semester_head, Course* &course_head).**



```
------------Nhap Mon Lap Trinh_Semester1_2022-2023--------------

PLEASE INPUT THE NAME OF THE FILE YOU WANT TO UPDATE.
THE NAME OF THE FILE SHOULD BE CourseID_SemesterOrd_Year (CSC10002_Semester2_2019-2020_Scoreboard.csv)

ENTER FILE NAME:
```

Figure II.3.h.v.1: Input name of scoreboard file



```
------------Nhap Mon Lap Trinh_Semester1_2022-2023--------------

PLEASE INPUT THE NAME OF THE FILE YOU WANT TO UPDATE.
THE NAME OF THE FILE SHOULD BE CourseID_SemesterOrd_Year (CSC10002_Semester2_2019-2020_Scoreboard.csv)
ERROR: UNABLE TO OPEN FILE FOR READING

ENTER FILE NAME:
scoreboard_CSC10001_22CLC02.csv
```

Figure II.3.h.v.2: Imported successfully

i. Add a new class

void addNewClass (Year* &year_head, string username): This function adds a new class to a linked list of classes. The user is prompted to input the name of the new class and the input is validated to ensure it is not an empty string and doesn't already exist in a text file. If the input is valid, a new Class object is created and added to the linked list, and the name of the new class is appended to the text file (class.txt) while also creates a new file string name_file = new_class->class_name + "-" + year_head->year_name + ".txt". If the input is invalid, the user is prompted to input the name of the new class again. The function also includes various console commands for displaying the interface and receiving input from the user.

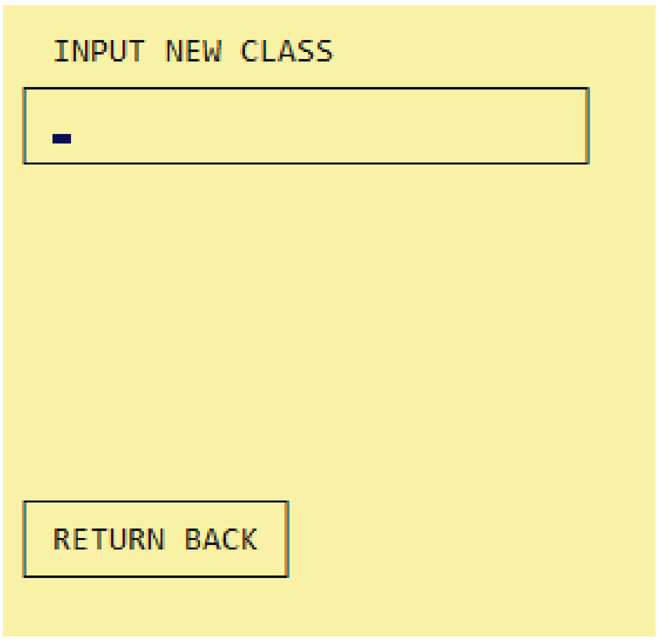
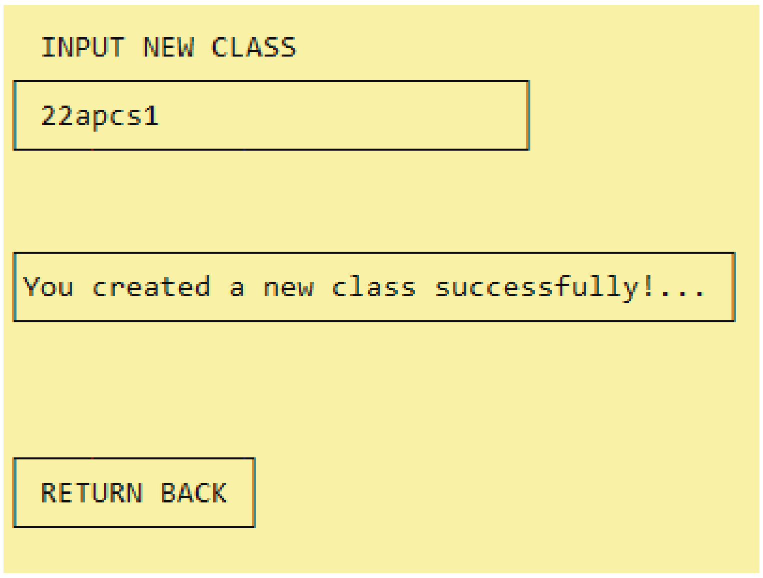


Figure II.3.i.1: Input the name of Class



```
22apcs1
```

```
You created a new class successfully!...
```

Figure II.3.i.2: Created a class successfully

j. Access a class

int CLASS_Interface(string &username, Year* &year_head, Class *class_head): This function displays all the options in a formatted way for the user to manage efficiently with the following choices: "IMPORT NEW STUDENT TO .CSV FILE", "VIEW ALL STUDENTS IN THIS CLASS", "VIEW SCOREBOARD FOR THIS CLASS", "RETURN BACK". Each of these choices corresponds to a different integer value, allowing for easy selection by the user.

void accessClass(string username, Year* &year_head, Class *class_head): This function takes the integer value returned through CLASS_Interface function and call the appropriate function for each and every option.

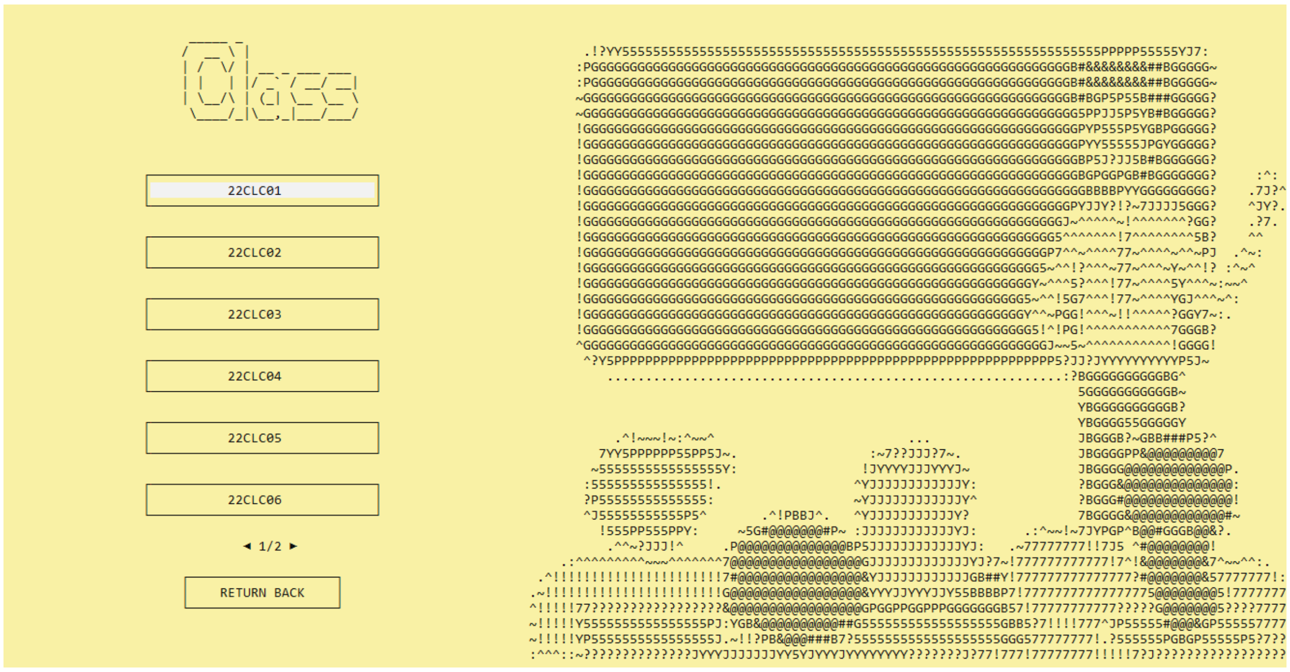


Figure II.3.j.1: The screen for viewing all created classes

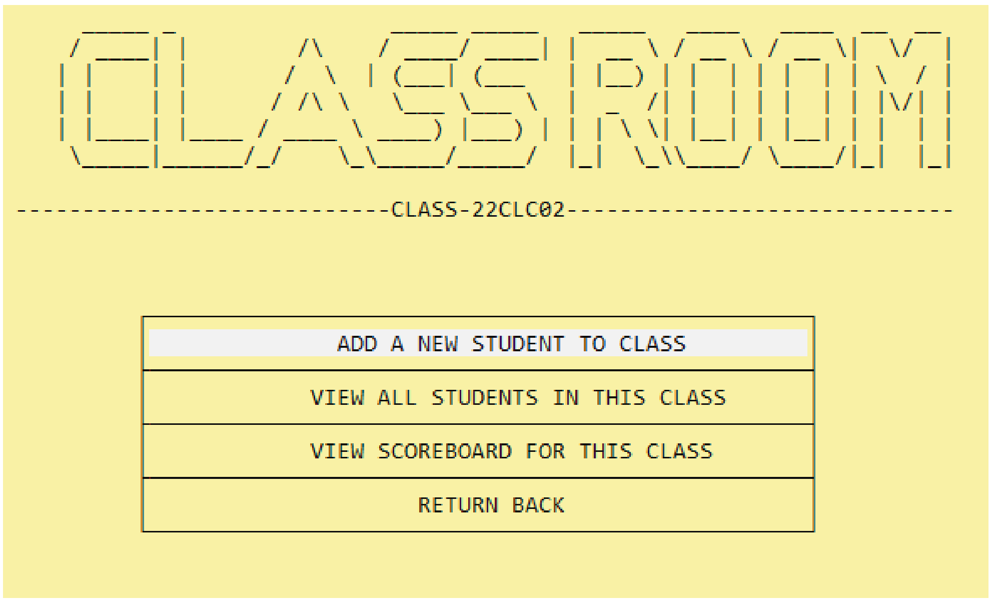

CLASSROOM
---------------------------CLASS-22CLC02---------------------------
ADD A NEW STUDENT TO CLASS
VIEW ALL STUDENTS IN THIS CLASS
VIEW SCOREBOARD FOR THIS CLASS
RETURN BACK

Figure II.3.j.2: Options in accessClass Screen

i. Add A New Student

void addNewStudent(string username, Year* &year_head, Class* cur): This function adds a new student to a class by appending their information to a file (***string filename* = *cur->class_name* + "-" + *year_head->year_name* + ".txt"**) in a specific format. If the file cannot be opened, an error message is displayed. Once the student information is added, a success message is displayed along with the option to return to the previous menu.

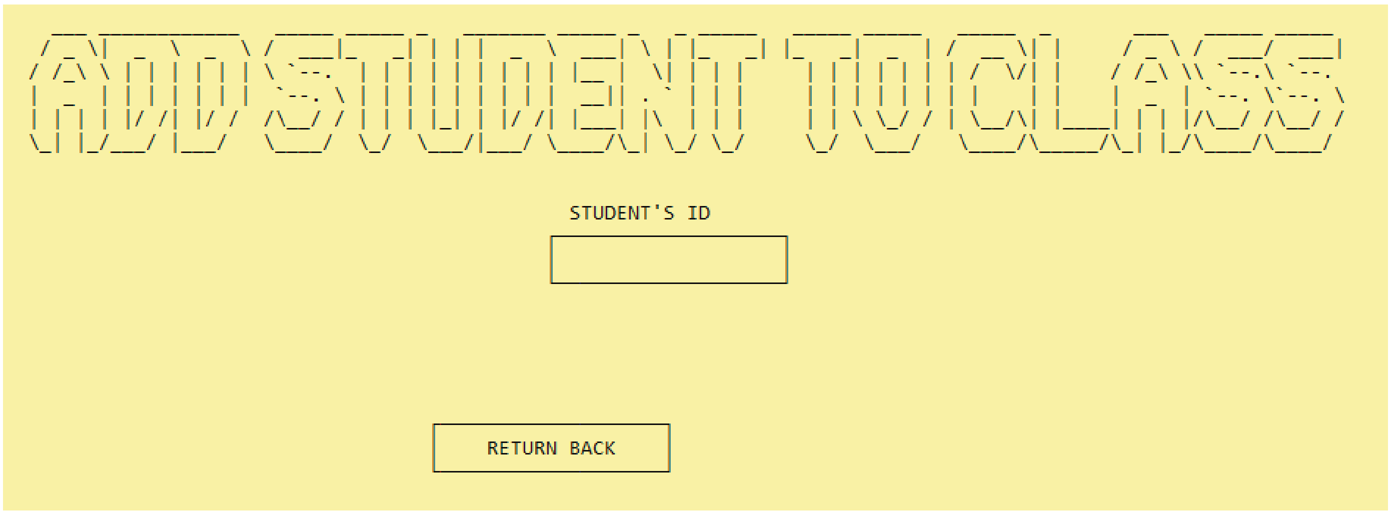


Figure II.3.j.i.1: Input student ID to add student

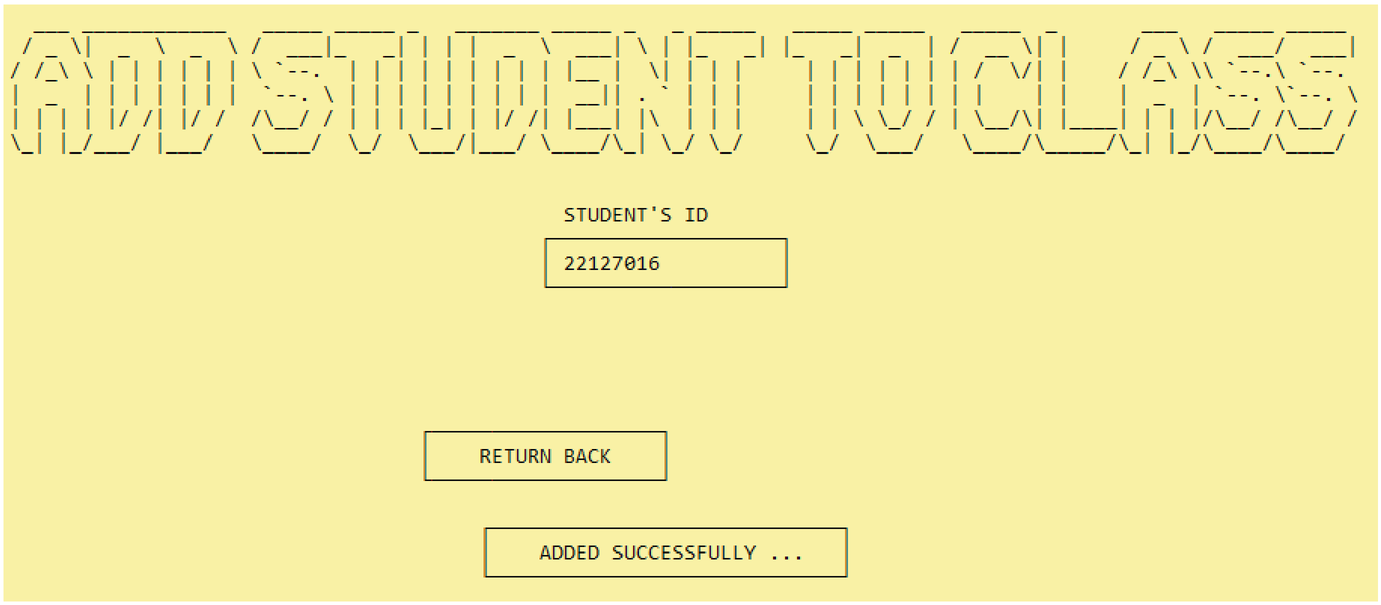
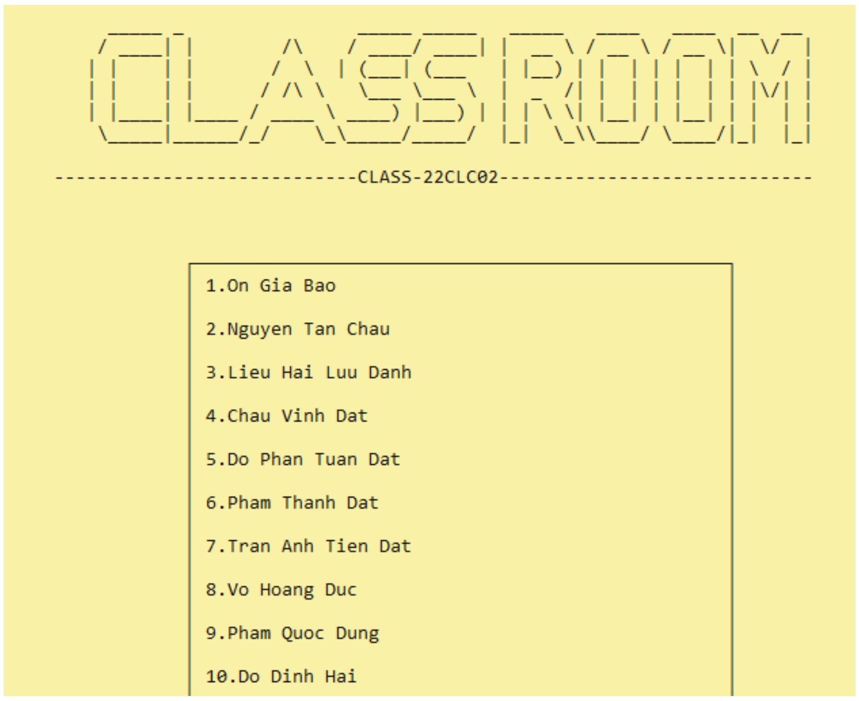


Figure II.3.j.i.2: added successfully

ii. View Students in Class

void viewStudentInClass(string username, Year* &year_head, Class *class_head): This function is used to display a list of students in a class. If there are no students, it displays a message saying so. Otherwise, it creates a box and fills it with a numbered list of the students' first and last names. It uses a while loop to iterate through the linked list that contains each student in the class and print their name in the correct format.



```
---------------------------CLASS-22CLC02---------------------------

1.On Gia Bao
2.Nguyen Tan Chau
3.Lieu Hai Luu Danh
4.Chau Vinh Dat
5.Do Phan Tuan Dat
6.Pham Thanh Dat
7.Tran Anh Tien Dat
8.Vo Hoang Duc
9.Pham Quoc Dung
10.Do Dinh Hai
```

Figure II.3.j.ii.1: The list of students in a class

iii. View Scoreboard of a Class

int getSemesterNum(): The function provides a user interface to select one out of the three semesters, namely the first, second, and third semesters, and returns the selected semester value.

void viewScoreBoard_Class(string username, Year *&year_head, Class *class_head): This function gets value from **getSemesterNum()**, before displaying students with the same format and uses the same helper functions as the **viewScoreBoard_Course** function and takes information of courses ***from filename = class_head->class_name+ "_" + "Semester" + char_semester + "_" + year_head->year_name + "_courses.csv"*** and the scoreboard information from files with the format ***filename1 = check->course_ID + "_Semester" + char_semester + "_" + year_head->year_name + "_Scoreboard.csv"***.

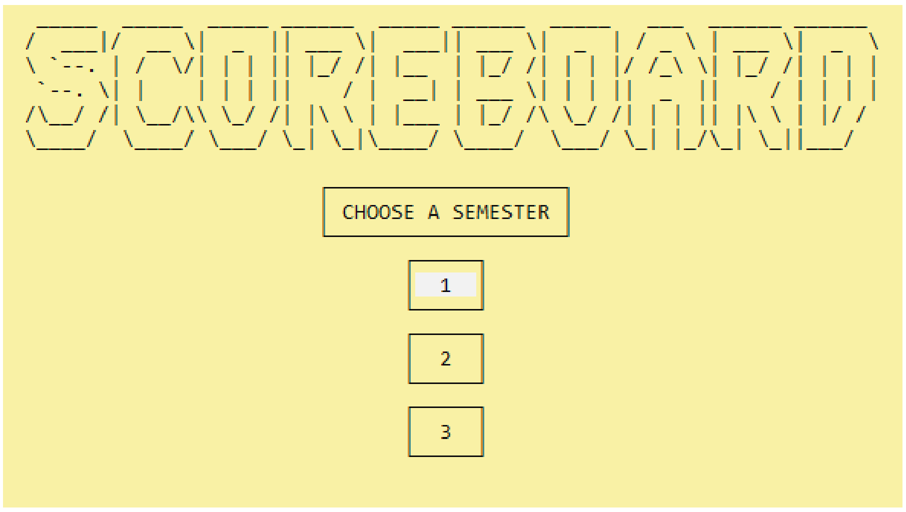
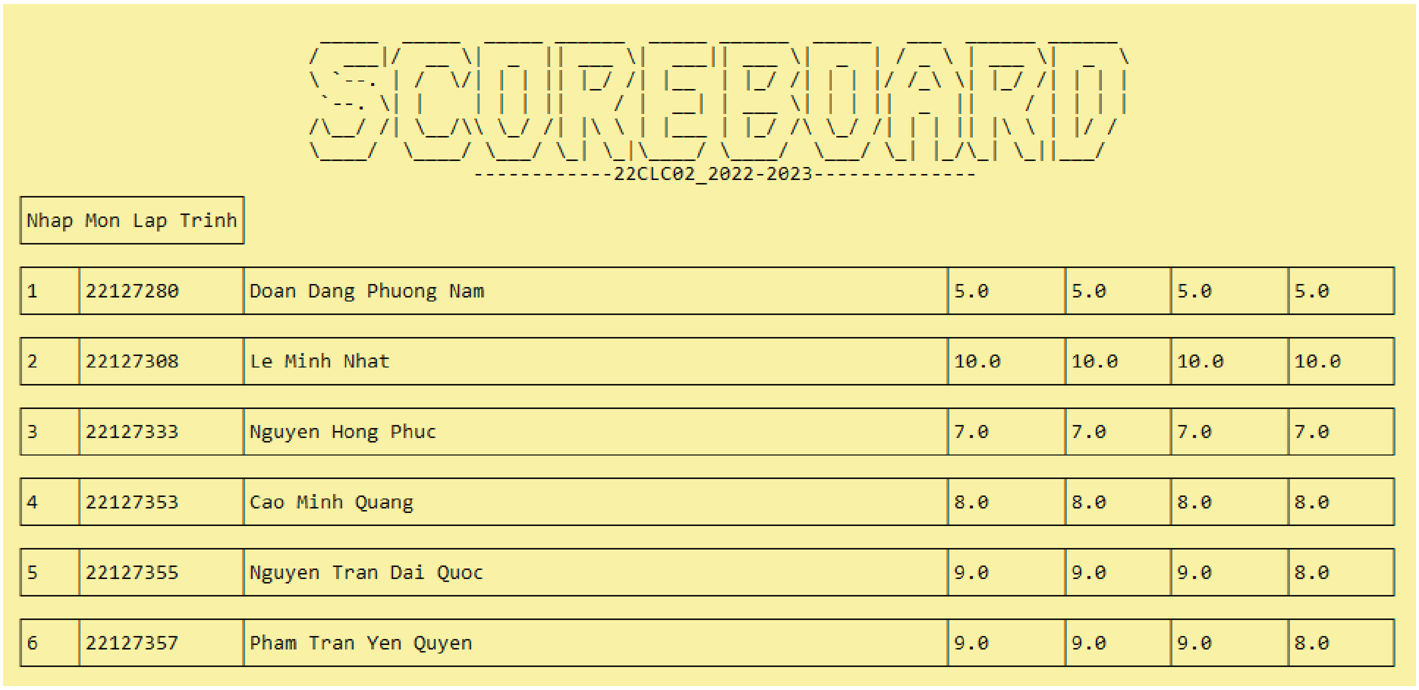


Figure II.3.j.iii.1: Choose a semester to view scoreboard of class



```
------------22CLC02_2022-2023-------------
```

Nhap Mon Lap Trinh

1	22127280	Doan Dang Phuong Nam	5.0	5.0	5.0	5.0
2	22127308	Le Minh Nhat	10.0	10.0	10.0	10.0
3	22127333	Nguyen Hong Phuc	7.0	7.0	7.0	7.0
4	22127353	Cao Minh Quang	8.0	8.0	8.0	8.0
5	22127355	Nguyen Tran Dai Quoc	9.0	9.0	9.0	8.0
6	22127357	Pham Tran Yen Quyen	9.0	9.0	9.0	8.0

Figure II.3.j.iii.2: The scoreboard of a class

4. Student

a. Main Screen

void Main_Student_Screen(string username): This function creates an interactive menu for students. It presents five different options, including **"LOG OUT", "VIEW PROFILE", "VIEW COURSES", "VIEW SCOREBOARD", and "CHANGE PASSWORD"**, each displayed in its own stylized interface element. Users can navigate the menu by using arrow keys, and the currently selected option is highlighted. Once an option is selected, the function will return a value corresponding to that option. To provide a cleaner interface, the function hides the cursor like all the other above mentioned functions.

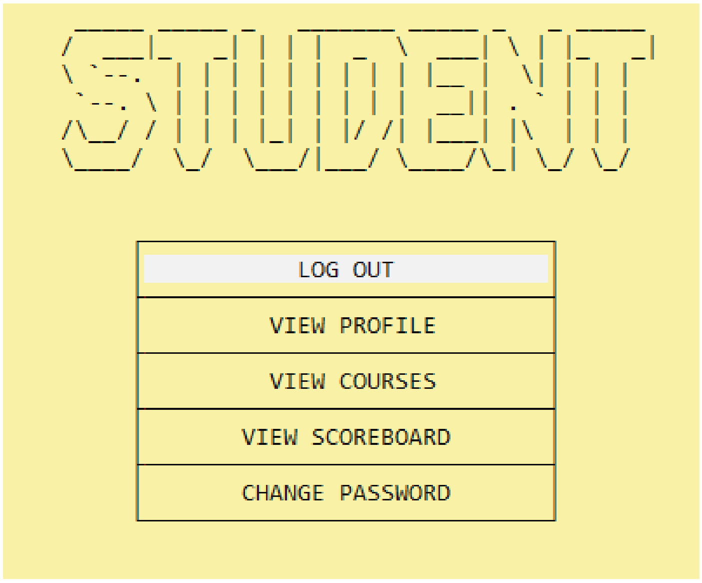
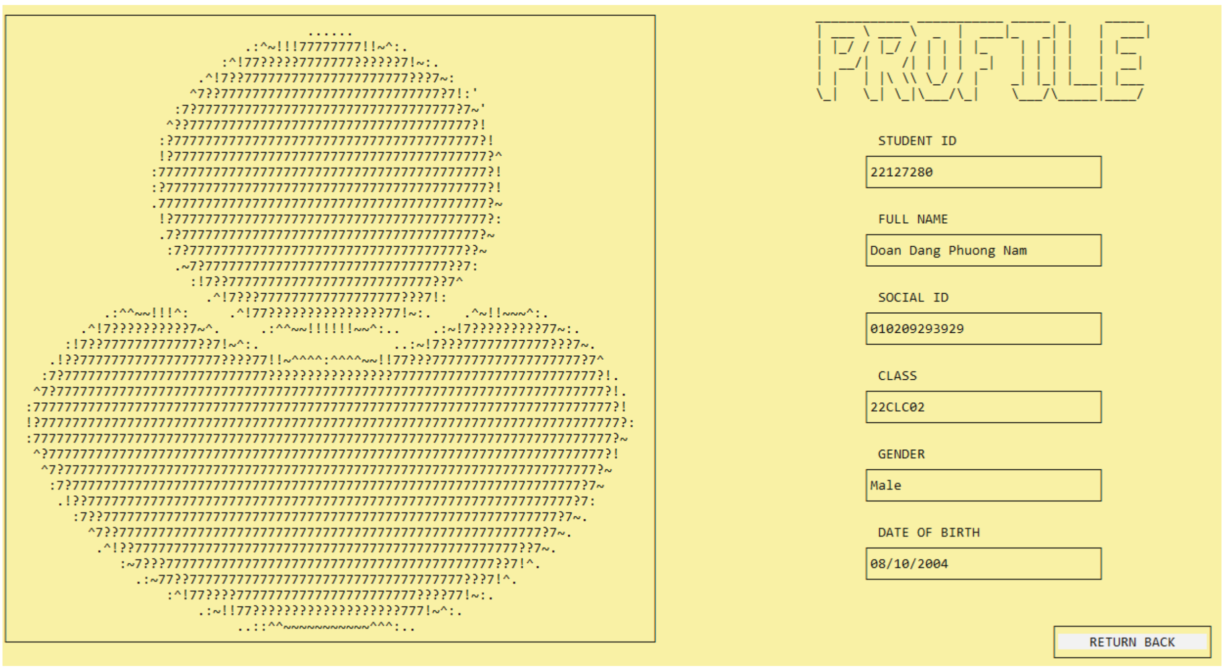


Figure II.4.a.1: Main Student Screen

void student_info(string username): This functions is similar to staff_info but instead access **student_info.csv** to find the appropriate values.



22127280

Doan Dang Phuong Nam

010209293929

22CLC02

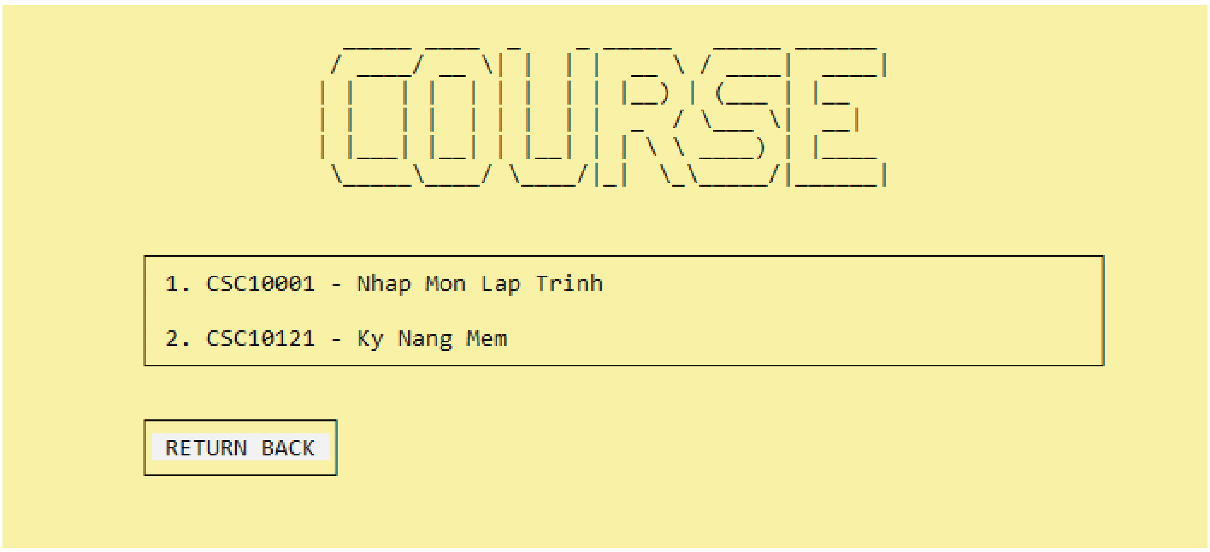
Male

08/10/2004

Figure II.4.a.2: Student's profile

b. View Student's Courses

void viewCoursesInSemester(string username): This function displays the courses a student is taking in the current semester. It retrieves student information from ***student_info.csv***, gets the current semester and year using previously mentioned helper functions, and reads course information from ***filename = user->student_class.class_name + "_Semester" + ch_semester + "_" + year + "_courses.csv"***. It then checks if the student is enrolled in each course by reading multiple files with the same syntax ***filename1 = check->course_ID + "_Semester" + ch_semester + "_" + check->class_name + "_" + year + "_student.csv"***, and displays the courses the student is enrolled in.



```
1. CSC10001 - Nhap Mon Lap Trinh

2. CSC10121 - Ky Nang Mem
```

Figure II.4.b.1: Student's courses

c. View Student's Scoreboard

void viewScoreBoard_Student(string username): This function reads a student's information from a CSV file and prompts the user for a semester and year. It then **displays the student's score** for each course they are enrolled in for that semester and year by looping through each course and reading from a CSV file containing the course's score information.

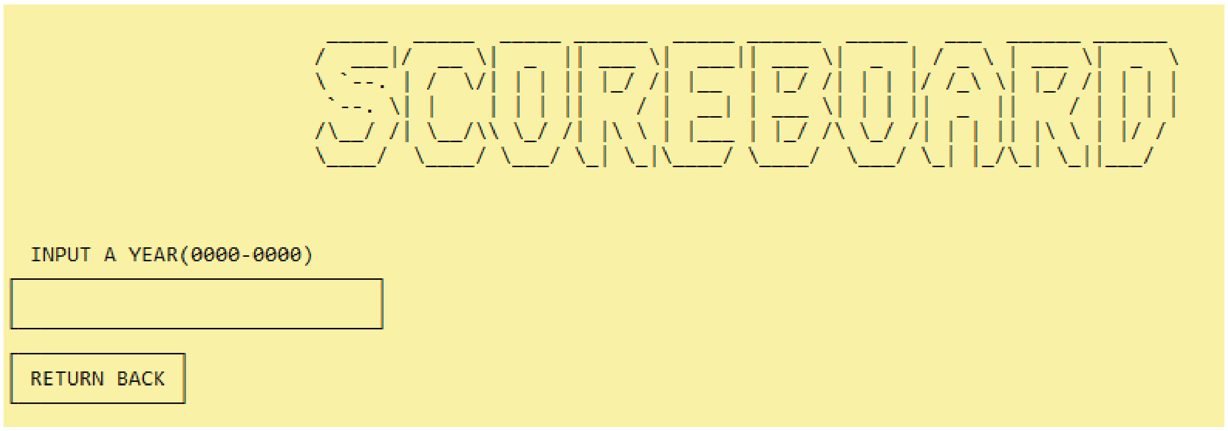


Figure II.4.c.1: Input year for view Student's scoreboard

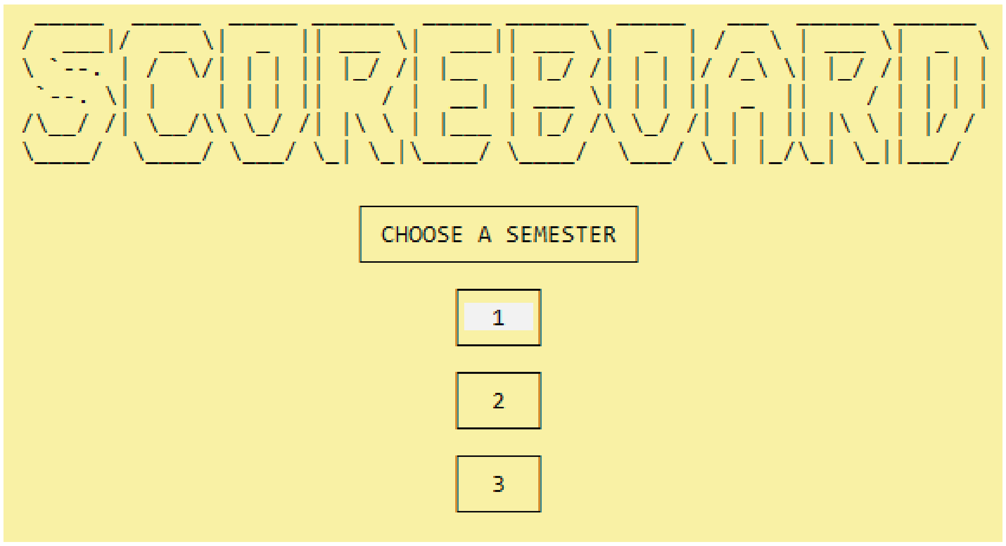
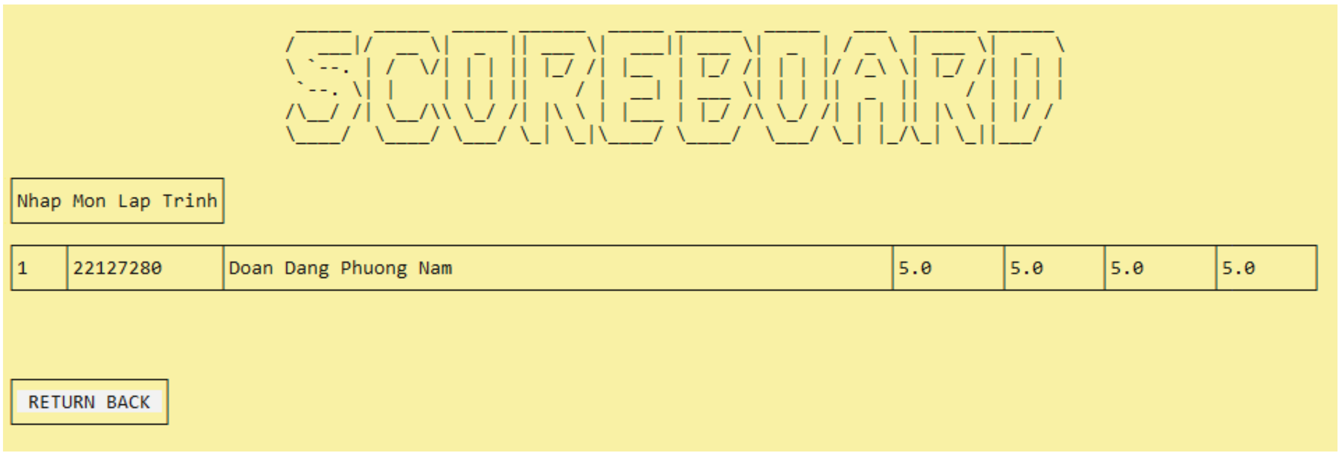


Figure II.4.c.2: Choose a semester



1	22127280	Doan Dang Phuong Nam	5.0	5.0	5.0	5.0

Figure II.4.c.3: Student's scoreboard

III. Advance Features

1. Visual Effects

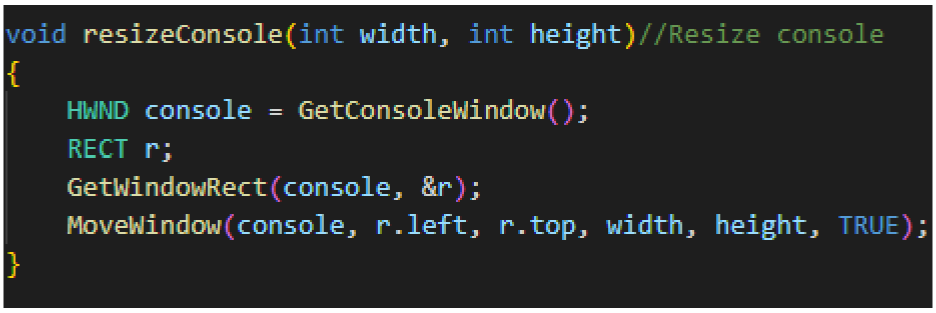
This is one of the essential parts of the program which provides us with a user interface. To add some stunning visual effects to my console ìnterface, we included **Windows.h** library, which can assist us to design boxes for choosing options, as well as some features like moving cursor to any positions, hiding/unhiding cursor or setting the size of screen.

Talking a little bit about **Windows.h** library, it is a C/C++ header file that contains declarations for all of the functions, structures, and constants needed for programming Windows applications. It provides the necessary definitions for creating Windows applications, including the creation of windows, menus, and dialog boxes, as well as handling input, output, and event notifications.

In this report, we will present about 4 main effects which were used frequently in the source code of our group.

a. Resize Console Screen

To resize the width and the height of console screen, our group wrote a function called **resizeConsole().** The image below is our function:



```
void resizeConsole(int width, int height)//Resize console
{
    HWND console = GetConsoleWindow();
    RECT r;
    GetWindowRect(console, &r);
    MoveWindow(console, r.left, r.top, width, height, TRUE);
}
```

Figure III.1.a.1: Function to resize console screen

Depending on the setting of user’s screen, two parameters *width* and *height* will be different.

b. Create A Box

Before discussing about how to make an even and attractive box, we will briefly look into a function that help us to move cursor to any positions in a console interface, which is **gotoXY.**

A console screen is considered a coordinate system with two axises Ox and Oy like the below figure.

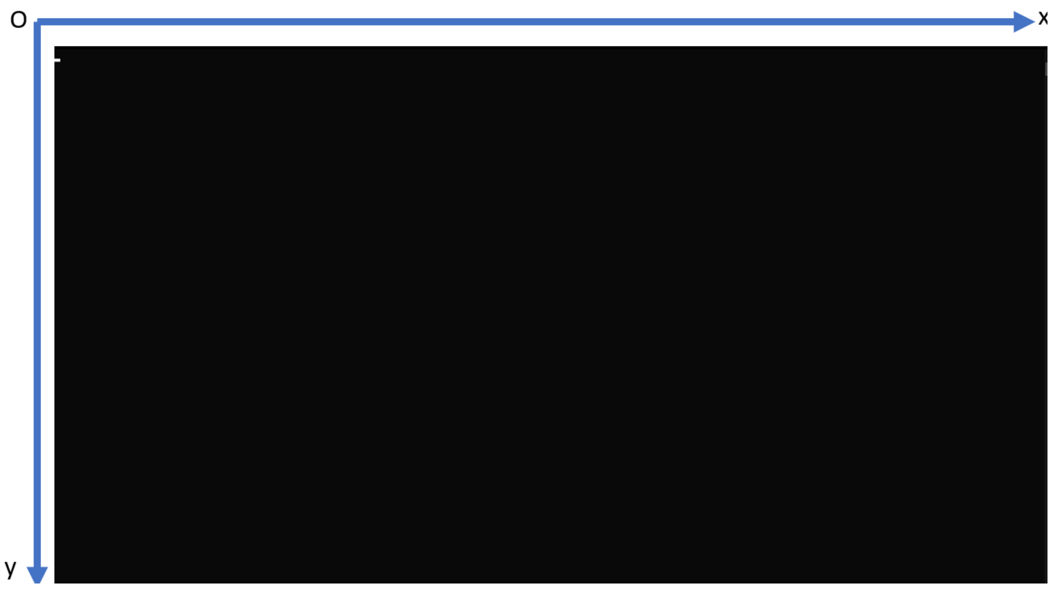
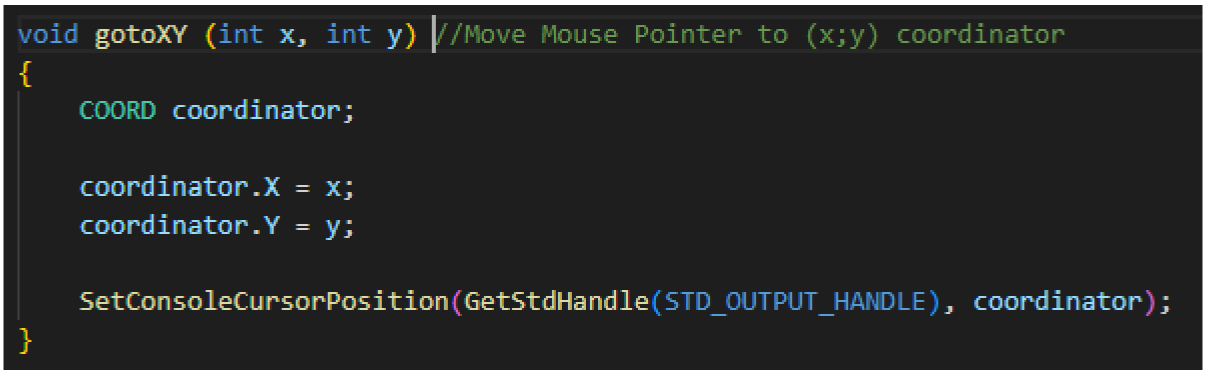


Figure III.1.b.1: A console screen

Hence, the function **gotoXY** allows us to move cursor to a point (x;y). We will write this function:



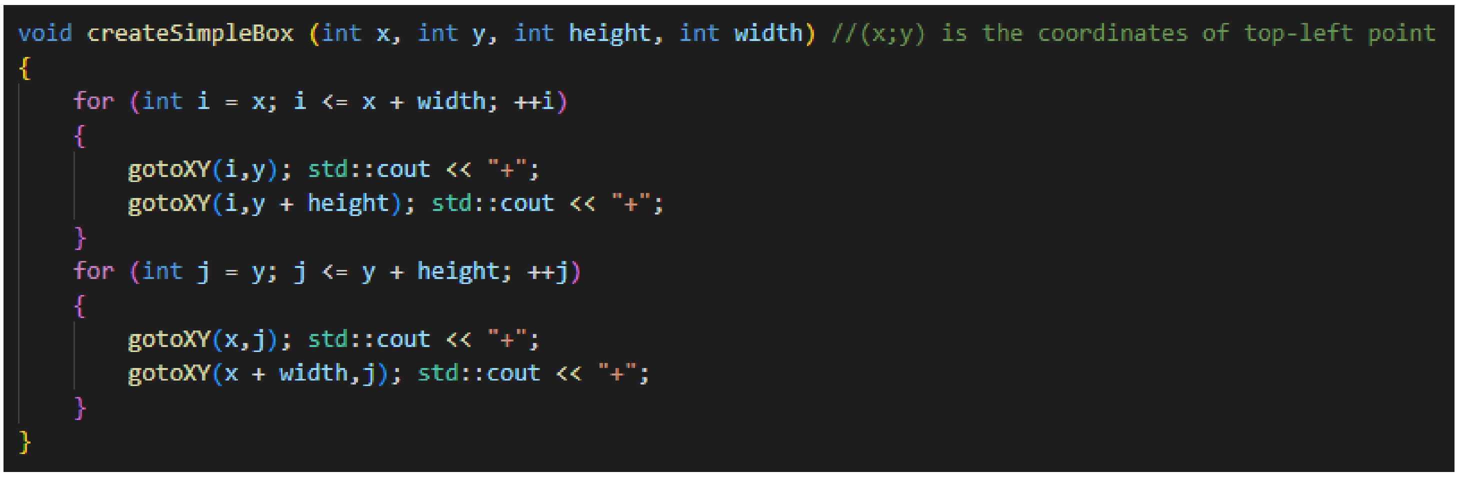
```
void gotoXY (int x, int y) //Move Mouse Pointer to (x;y) coordinator
{
    COORD coordinator;

    coordinator.X = x;
    coordinator.Y = y;

    SetConsoleCursorPosition(GetStdHandle(STD_OUTPUT_HANDLE), coordinator);
}
```

Figure III.1.b.2: Implementation with the function **gotoXY**

After exploring the function **gotoXY**, we will move to main content, which is designing a uniform rectangle (called a box) on a screen. A rectangle is a shape with 4 right angles, having the width and the height, as a result, to obtain a box, we need to define the size of width and height. Then, we will design on a fundamental concept:

On a horizontal line, with each point (x:y), I will mark it by a letter and after that, I will continue to express the point (x; y + h) with the same letter (h is the symbol of the height). Vertical line is similar.

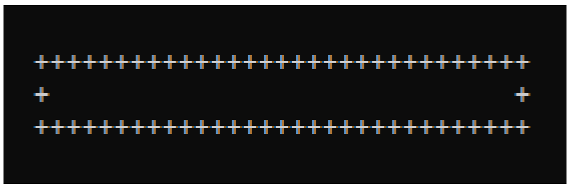
Based on that idea, I have a source code:



```
void createSimpleBox (int x, int y, int height, int width) //(x;y) is the coordinates of top-left point
{
    for (int i = x; i <= x + width; ++i)
    {
        gotoXY(i,y); std::cout << "+";
        gotoXY(i,y + height); std::cout << "+";
    }
    for (int j = y; j <= y + height; ++j)
    {
        gotoXY(x,j); std::cout << "+";
        gotoXY(x + width,j); std::cout << "+";
    }
}
```

Figure III.1.b.3: Create a simple box with full "+"

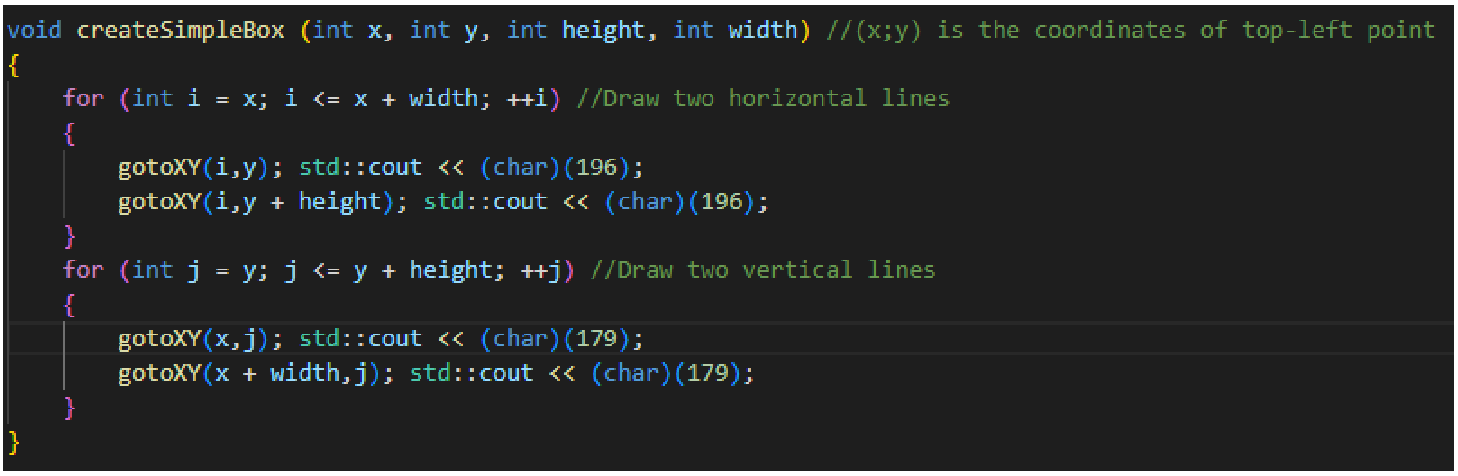
And this is the created box:



```
+++++++++++++++++++++++++++++++
+                             +
+++++++++++++++++++++++++++++++
```

Figure III.1.b.4: A "+" rectangle

To update it by change "+" to another letters which look like a line symbol, we found on ASCII Table and saw two corresponding letters: 179(|) and 196(-). Now I will update my function.



```
void createSimpleBox (int x, int y, int height, int width) //(x;y) is the coordinates of top-left point
{
    for (int i = x; i <= x + width; ++i) //Draw two horizontal lines
    {
        gotoXY(i,y); std::cout << (char)(196);
        gotoXY(i,y + height); std::cout << (char)(196);
    }
    for (int j = y; j <= y + height; ++j) //Draw two vertical lines
    {
        gotoXY(x,j); std::cout << (char)(179);
        gotoXY(x + width,j); std::cout << (char)(179);
    }
}
```

Figure III.1.b.5: Updated source code

And this is our box:

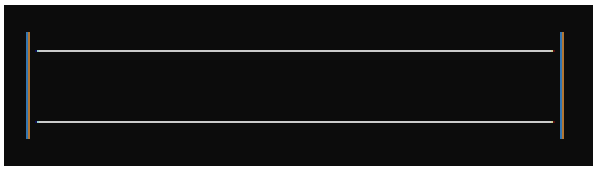
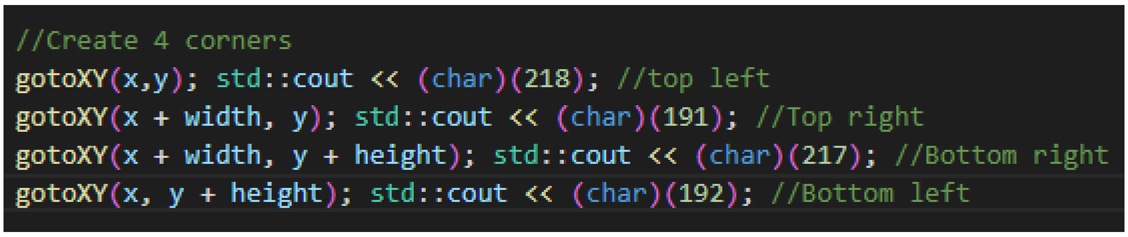


Figure III.1.b.6: Updated box

Currently, it seems to be missed 4 corners, to fix this, we return to ASCII Table and use 4 letters related to four corners:



```
//Create 4 corners
gotoXY(x,y); std::cout << (char)(218); //top left
gotoXY(x + width, y); std::cout << (char)(191); //Top right
gotoXY(x + width, y + height); std::cout << (char)(217); //Bottom right
gotoXY(x, y + height); std::cout << (char)(192); //Bottom left
```

Figure III.1.b.7: Source code for designing 4 corners

And this is our even box:

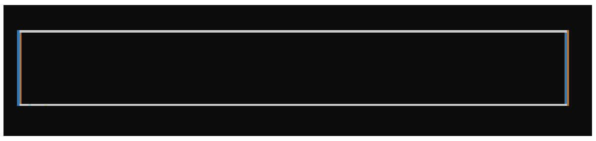
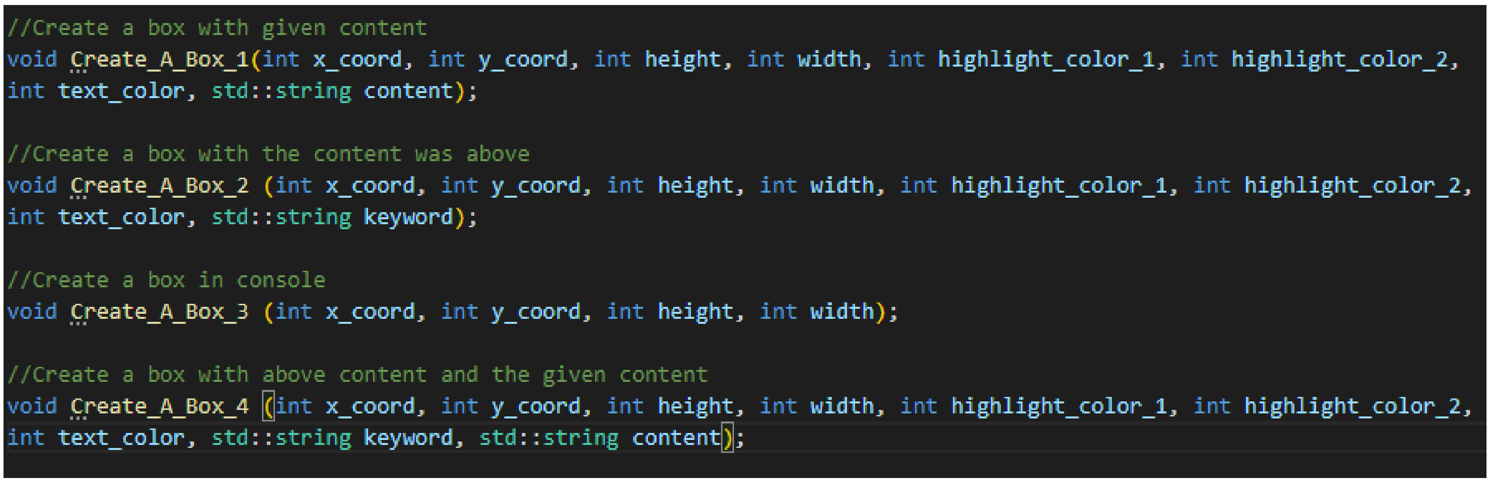


Figure III.1.b.8: A rectangle box

In our group project, we create four kinds of box which were used according to the aim of each group member.

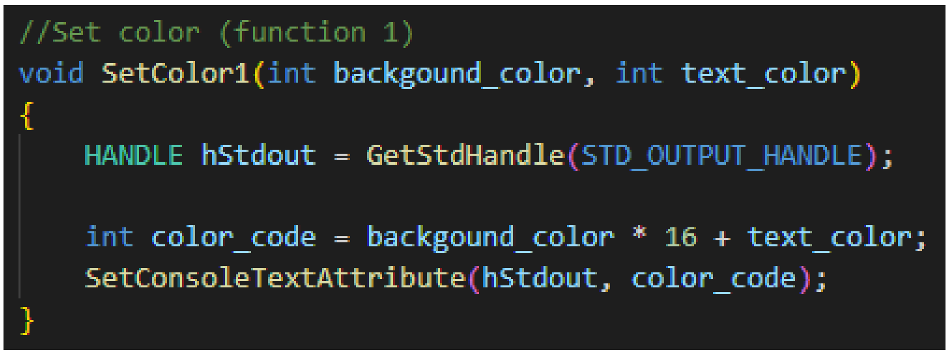


```
//Create a box with given content
void Create_A_Box_1(int x_coord, int y_coord, int height, int width, int highlight_color_1, int highlight_color_2,
int text_color, std::string content);

//Create a box with the content was above
void Create_A_Box_2 (int x_coord, int y_coord, int height, int width, int highlight_color_1, int highlight_color_2,
int text_color, std::string keyword);

//Create a box in console
void Create_A_Box_3 (int x_coord, int y_coord, int height, int width);

//Create a box with above content and the given content
void Create_A_Box_4 (int x_coord, int y_coord, int height, int width, int highlight_color_1, int highlight_color_2,
int text_color, std::string keyword, std::string content);
```

Figure III.1.b.9: Four prototypes for four kinds of box

c. Color Effects

To create a function that modifies the font or background color in the console screen, we continue to include **Windows.h** library. In our group project, we write the function SetColor1() to change the text and color on a line of console screen.



```
//Set color (function 1)
void SetColor1(int backgound_color, int text_color)
{
    HANDLE hStdout = GetStdHandle(STD_OUTPUT_HANDLE);

    int color_code = backgound_color * 16 + text_color;
    SetConsoleTextAttribute(hStdout, color_code);
}
```

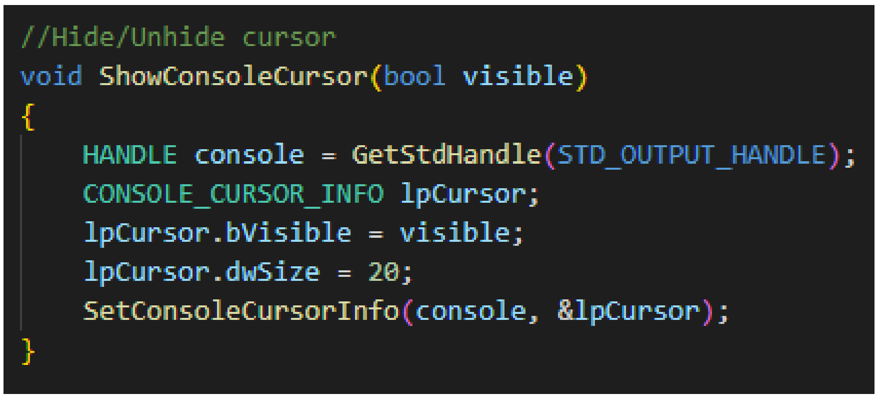
Figure III.1.c.1: Code to change color

Background_color and *text_color* will be an integer from 0 to 15, and the table below is the convention of color with corresponding numbers.

Integer	Color
0	Black
1	Navy
2	Green
3	Teal
4	Maroon
5	Purple
6	Olive
7	Silver
8	Gray
9	Blue
10	Lime
11	Aqua
12	Red
13	Fuchsia
14	Yellow
15	White

d. Hide/Unhide Cursor

To hide/unhide cursor in the console screen, we need a function called **ShowConsoleCursor().** If *visible* is **false**, the cursor will be unhided, in contrast, its hiding status will be stimulated.

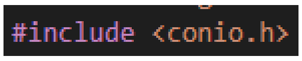


```
//Hide/Unhide cursor
void ShowConsoleCursor(bool visible)
{
    HANDLE console = GetStdHandle(STD_OUTPUT_HANDLE);
    CONSOLE_CURSOR_INFO lpCursor;
    lpCursor.bVisible = visible;
    lpCursor.dwSize = 20;
    SetConsoleCursorInfo(console, &lpCursor);
}
```

Figure III.1.d.1: A function for modifying the cursor status

2. Implementation with user's input

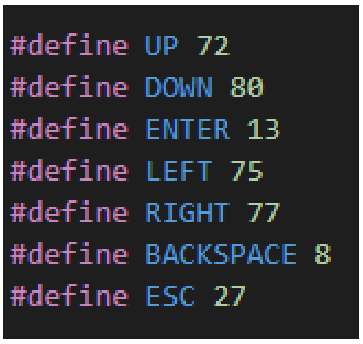
In the project, when we worked with **<Windows.h>** library for a period of time, we realized that if we had just used **std::cin** >> to get input from users, we will not to be able to implement anything except waiting for users' input, while we want to allow them to return to the previous page at any time. Hence, we found a solution for that, which is including a library **<Conio.h>**



```
#include <conio.h>
```

Figure III.2.1: **<Conio.h>** library

After including this library, we used two functions which are **_getch()** and **_kbhit()** on a regular basis. With **_kbhit()**, when users press any keys on their keyboard, **_kbhit()** will return **true** and then, we will assign that key to a character variable (supposes it is called **c**) by typing **c = _getch()**. Here are some typical keys which was commonly used by our group.



```
#define UP 72
#define DOWN 80
#define ENTER 13
#define LEFT 75
#define RIGHT 77
#define BACKSPACE 8
#define ESC 27
```

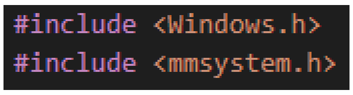
Figure III.2.2: typical keys

For inputs containing the characters M and K, since these characters coincide with the LEFT and RIGHT characters in the ASCII character set, when inputting, please enter "m" and "k", and the system will automatically capitalize these 2 characters for the user.

3. Sound Effects

Our group made the project on Visual Studio Code, so in the content of the report, we will present about how to add and compile sound effects on this IDE.

Firstly, if we want to be able to use some audio support functions from C++, we need to include **<mmsystem.h>** and **<Windows.h>** library, since the implementation part belongs to the winmm.lib library into the Linker section, without this library there will be no executable code for audio functions.



```
#include <Windows.h>
#include <mmsystem.h>
```

Figure III.3.1: Two libraries for adding audio

Then, we use a function **PlaySound()** (we can change to **PlaySoundA()**) to add sounds.

bool PlaySound(LPCWSTR FileName, HMODULE TypeOfSound, DWORD Flag);

The above function will return true if the audio is playable and vice versa.

fileName: path to the WAV audio file to be played.

TypeOfSound: normally will be NULL if an audio file is in WAV format.

Flag: flag to play an audio file, here use:

• SND_FILENAME: if there is this parameter, the first parameter will be a string leading to the audio file

• SND_LOOP: repeat sound

• SND_ASYNC: music runs in the background, to turn off the music, call this function again but the first parameter is NULL.

• SND_NOSTOP: The sound will be played until there is a command requiring the end of sound.

For instance, we have an audio file WAV called **BackgroundMusic.wav**, consequently, in our source code, we will add a command line:

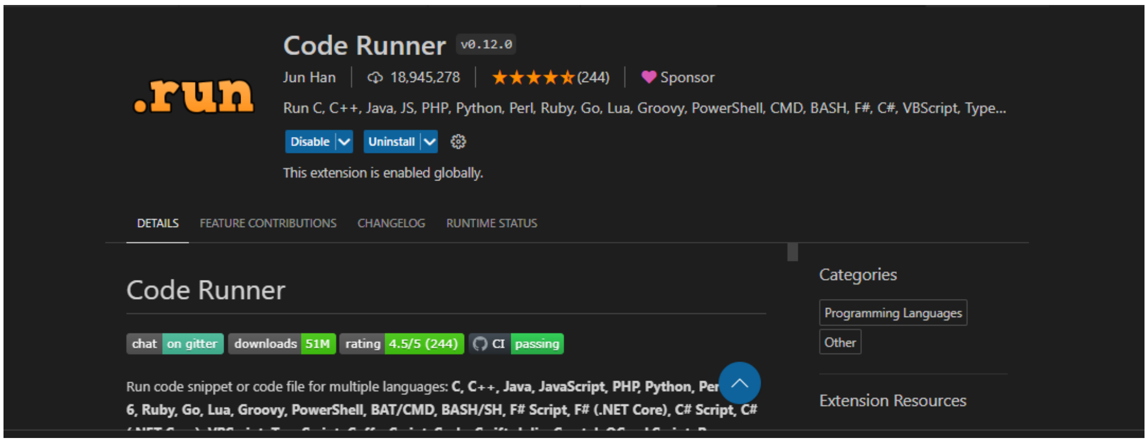
```
PlaySoundA("BackgroundMusic.wav", NULL, SND_FILENAME | SND_ASYNC | SND_LOOP | SND_NOSTOP);
```

Or

```
PlaySound(TEXT("BackgroundMusic.wav"), NULL, SND_FILENAME | SND_ASYNC | SND_LOOP | SND_NOSTOP);
```

In **Visual Studio Code**, to compile the audio functions, according to the agreement of the group, we decided to implement 5 steps:

Step 1: Install **Code Runner** extension



Code Runner v0.12.0

Jun Han | 18,945,278 | ★★★★★(244) | Sponsor

Run C, C++, Java, JS, PHP, Python, Perl, Ruby, Go, Lua, Groovy, PowerShell, CMD, BASH, F#, C#, VBScript, Type...

This extension is enabled globally.

Code Runner

chat on gitter | downloads 51M | rating 4.5/5 (244) | CI passing

Run code snippet or code file for multiple languages: **C, C++, Java, JavaScript, PHP, Python, Per**
6, Ruby, Go, Lua, Groovy, PowerShell, BAT/CMD, BASH/SH, F# Script, F# (.NET Core), C# Script, C#

Categories

Programming Languages

Other

Extension Resources

Figure III.3.2: Code Runner

Step 2: In **Settings**, type the keyword **"Code Runner"**.

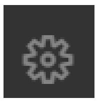
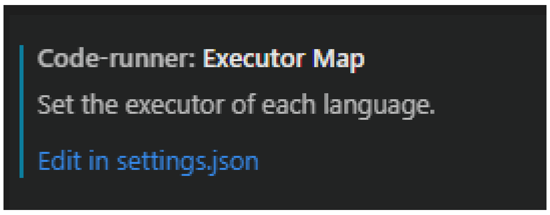


Figure III.3.3: Setting symbol



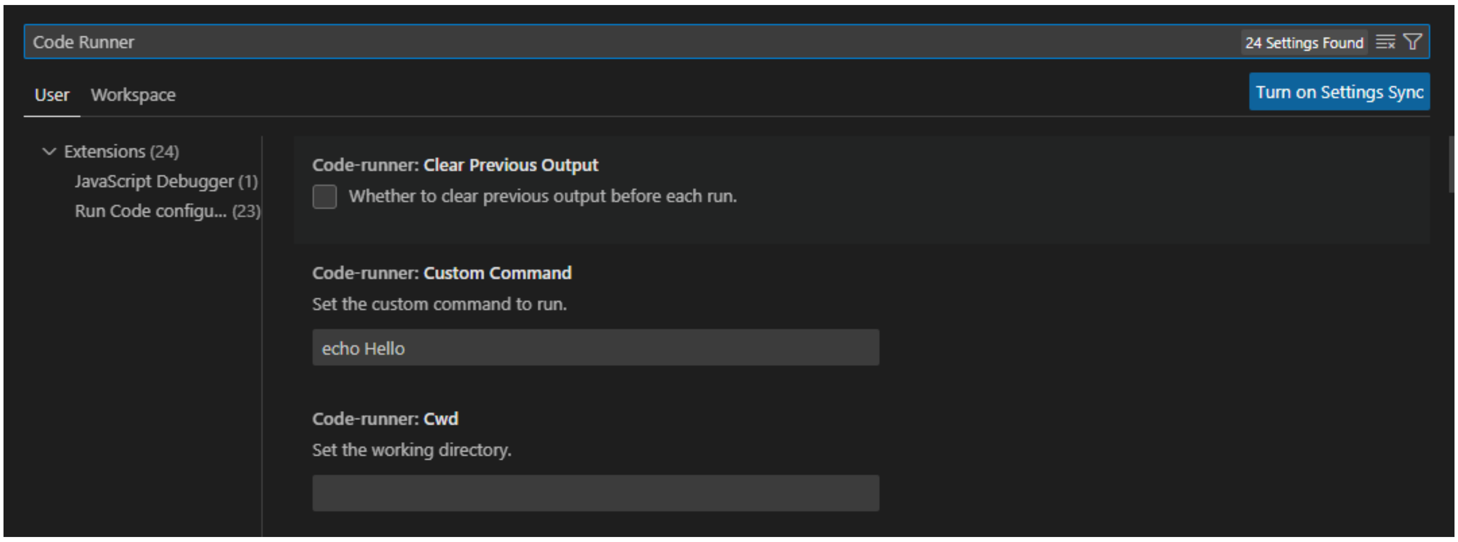
Code-runner: Executor Map

Set the executor of each language.

Edit in settings.json

Figure III.3.4: Type “Code Runner” in Settings

Step 3: Move to **Executor Map** and choose Edit in **Settings.json**.



Code-runner: Clear Previous Output

Whether to clear previous output before each run.

Code-runner: Custom Command

Set the custom command to run.

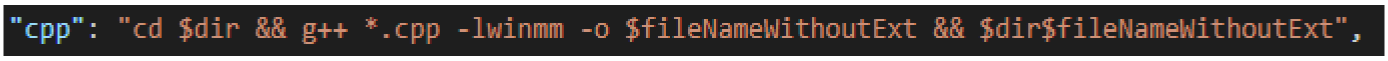
echo Hello

Code-runner: Cwd

Set the working directory.

Figure III.3.5: Executor Map

Step 4: Go to **“cpp”** and follow the image below.



```
"cpp": "cd $dir && g++ *.cpp -lwinmm -o $fileNameWithoutExt && $dir$fileNameWithoutExt",
```

Figure III.3.6: Adding -lwinmm

Step 5: Compile all cpp files by touching key combinations: **Ctrl + Atl + N**.

References

[1] ASCII Image from ASCII Art Archive,

Link: https://www.text-image.com/convert/ascii.html

[2] Art Font Size from ASCII Art Archive Text,

Link: https://patorjk.com/software/taag/#p=display&f=Graffiti&t=Type%20Something%20

[3] Concept for building Dynamic Menu,

Link: https://www.youtube.com/watch?v=oDh046cT_Q0&t=2s

[4] Knowledge about <Windows.h> library,

Link

1. https://codelearn.io/sharing/windowsh-va-ham-dinh-dang-console-p1
2. https://codelearn.io/sharing/windowsh-ham-dinh-dang-noi-dung-console
3. https://learn.microsoft.com/en-us/windows/win32/api/winbase/

[5] Soundtrack “Maybe”, Chill R&B Type Beat

Link: https://www.youtube.com/watch?v=IG4PXF1fkt8&feature=youtu.be