```
#Pateel Arabian
#CS008
#Project 1
#include <iostream>
#include <fstream>
#include <sstream>
#include <string>
using namespace std;

//Crerate student class to represent a students info
class Student {
private:
    int id;
    string name;

public:
    //setters
    Student(int studentId = 0, const string& studentName = "") :
id(studentId), name(studentName) {}

    //getters
    int getId() const { return id; }
    string getName() const { return name; }
};

// Define a node class for creating linked lists of students
class node {
private:
    Student data;
    node* link;

public:
    //setters
    node(const Student& init_data = Student(), node* init_link = NULL) :
data(init_data), link(init_link) {}

    //getters
    Student getData() const { return data; }
    node* getLink() const { return link; }
    void setLink(node* new_link) { link = new_link; }
};
```

```
// Define a Course class to represent courses, including enrolled and
waitlisted students
class Course {
private:
    string code;
    string title;
    int enrolled;
    node* eList;
    int waitlist;
    node* wList;

public:
    //setters
    Course(const string& courseCode = "", const string& courseTitle = "",
int initialEnrolled = 0, int initialWaitlist = 0)
        : code(courseCode), title(courseTitle), enrolled(initialEnrolled),
waitlist(initialWaitlist), eList(nullptr), wList(nullptr) {}

    //getters
    string getCode() const { return code; }
    string getTitle() const { return title; }
    int getEnrolled() const { return enrolled; }
    int getWaitlist() const { return waitlist; }
    node* getEnrolledList() const { return eList; }
    node* getWaitlistList() const { return wList; }

    // Function to add students to the enrolled
    // creates a student and adds them to the list and inciments the
enrolled count
    void addStudentToEnrolled(const Student& student) {
        node* newStudent = new node(student, eList);
        eList = newStudent;
        enrolled++;
    }

    // Functions to add students to the waitlist
    // creates a student and adds them to the list and inciments the
waitlist count
    void addStudentToWaitlist(const Student& student) {
        node* newStudent = new node(student, wList);
        wList = newStudent;
        waitlist++;
    }
```

```
// Function to remove students from the enrolled
void removeStudentFromEnrolled(const Student& student) {
    node* current = eList;
    node* previous = nullptr;

    while (current != nullptr) {
        if (current->getData().getId() == student.getId()) {
            if (previous == nullptr) {
                eList = current->getLink();
            } else {
                previous->setLink(current->getLink());
            }

            delete current;
            enrolled--;
            return;
        }
        previous = current;
        current = current->getLink();
    }
}

// Function to remove students from the waitlist
void removeStudentFromWaitlist(const Student& student) {
    node* current = wList;
    node* previous = nullptr;

    while (current != nullptr) {
        if (current->getData().getId() == student.getId()) {
            if (previous == nullptr) {
                wList = current->getLink();
            } else {
                previous->setLink(current->getLink());
            }

            delete current;
            waitlist--;
            return;
        }
        previous = current;
        current = current->getLink();
    }
    }

    // Custom iterator for iterating through the enrolled students of a
course.
    class enrolled_iterator {
    private:
        node* current;  // Pointer to the current node in the linked list

    public:
    // Constructor for the iterator, takes the starting node as input.
    enrolled_iterator(node* startNode) : current(startNode) {}

    // Inequality operator for comparing two iterators. Used for loop
termination.
    bool operator!=(const enrolled_iterator& other) const {
        return current != other.current;
    }

    // Dereference operator to access the student data that the iterator is
pointing to.
    Student operator*() const {
        return current->getData();
    }

    // Increment operator to move the iterator to the next node in the
linked list.
    enrolled_iterator& operator++() {
        current = current->getLink();
        return *this;
    }
    };


    // Custom iterator for iterating through the waitlisted students of a
course.
    class waitlist_iterator {
    private:
        node* current;  // Pointer to the current node in the linked list

    public:
    // Constructor for the iterator, takes the starting node as input.
    waitlist_iterator(node* startNode) : current(startNode) {}
    // Inequality operator for comparing two iterators. Used for loop
termination.
    bool operator!=(const waitlist_iterator& other) const {
        return current != other.current;
    }

    // Dereference operator to access the student data that the iterator is
pointing to.
    Student operator*() const {
        return current->getData();
    }

    // Increment operator to move the iterator to the next node in the
linked list.
    waitlist_iterator& operator++() {
        current = current->getLink();
        return *this;
    }
    };


    // Functions to get the begin and end iterators for enrolled and
waitlist
    enrolled_iterator enrolled_begin() { return enrolled_iterator(eList); }
    enrolled_iterator enrolled_end() { return enrolled_iterator(nullptr); }
    waitlist_iterator waitlist_begin() { return waitlist_iterator(wList); }
    waitlist_iterator waitlist_end() { return waitlist_iterator(nullptr); }
};

/// Function to load course information from a file and populate the
courseArray.
// Parameters:
//   filename: The name of the file containing course information.
//   courseArray: An array of Course objects to store the loaded courses.
//   courseCount: A reference to an integer that keeps track of the number
of courses loaded.
void loadCourses(const string& filename, Course* courseArray, int&
courseCount) {
    ifstream file(filename);  // Open the specified file for reading.

    // Check if the file was successfully opened.
    if (!file.is_open()) {
        cout << "Error: Unable to open courses file." << endl;
        return;
    }

    string line;
    // Read each line from the file.
    while (getline(file, line)) {
        istringstream iss(line);
        string code, title;
        int enrolled, waitlist;
        iss >> code >> title >> enrolled >> waitlist;

        // Create a new Course object with the loaded information.
        Course course(code, title, enrolled, waitlist);

        // Add the course to the courseArray and increment the courseCount.
        courseArray[courseCount++] = course;
    }

    file.close();  // Close the file after reading its contents.
}


// Function to load student enrollments from a file and update the
courseArray.
// Parameters:
//   filename: The name of the file containing student enrollments.
//   courseArray: An array of Course objects representing available
courses.
//   courseCount: The number of courses in the courseArray.
void loadEnrollments(const string& filename, Course* courseArray, int
courseCount) {
    ifstream file(filename);  // Open the specified file for reading.

    // Check if the file was successfully opened.
    if (!file.is_open()) {
        cout << "Error: Unable to open enrollment file." << endl;
        return;
    }

    string line;
    // Read each line from the enrollment file.
    while (getline(file, line)) {
        istringstream iss(line);
        int studentId;
        string studentName;
        iss >> studentId >> studentName;

        // Create a Student object with the loaded student information.
        Student student(studentId, studentName);

        string courseCode;
        while (iss >> courseCode) {
            // Find the corresponding course in the courseArray.
            for (int i = 0; i < courseCount; i++) {
                if (courseArray[i].getCode() == courseCode) {
                    // Check if the course is full or has space in the
waitlist.
                    if (courseArray[i].getEnrolled() < 10) {
                        courseArray[i].addStudentToEnrolled(student);
                    } else {
                        courseArray[i].addStudentToWaitlist(student);
                    }
                    break;  // Break the loop once the course is found.
                }
            }
        }
    }

    file.close();  // Close the file after reading its contents.
}


// Function to display the main menu
int displayMenu() {
    cout << "\n--------------------------------------------------";
    cout << "\n                     Menu                         ";
    cout << "\n--------------------------------------------------";
    cout << "\n1. View your registration";
    cout << "\n2. Course registration";
    cout << "\n3. Course cancellation";
    cout << "\n4. Print enrollment list including waitlist";
    cout << "\n5. Exit";
    cout << "\n ---> Select : ";

    int choice;
    cin >> choice;
    return choice;
}

// Function to view a student's course registrations and waitlist status.
// Parameters:
//   courseArray: An array of Course objects representing available
courses.
//   courseCount: The number of courses in the courseArray.
void viewRegistration(Course* courseArray, int courseCount) {
    int studentId;
    string studentName;

    cout << "Enter your ID: ";
    cin >> studentId;
    cout << "Enter your name: ";
    cin.ignore();
    getline(cin, studentName);

    int enrolledCount = 0;  // Counter for enrolled courses.
    int waitlistCount = 0;  // Counter for waitlisted courses.

    for (int i = 0; i < courseCount; i++) {
        // Initialize custom iterators for enrolled and waitlisted students
in the current course.
        Course::enrolled_iterator enrolledIterator =
courseArray[i].enrolled_begin();
        Course::waitlist_iterator waitlistIterator =
courseArray[i].waitlist_begin();

        // Iterate through the enrolled students in the current course.
        while (enrolledIterator != courseArray[i].enrolled_end()) {
            // Check if the current student matches the provided ID and
name.
            if ((*enrolledIterator).getId() == studentId &&
(*enrolledIterator).getName() == studentName) {
                cout << "(R) " << courseArray[i].getCode() << "    " <<
courseArray[i].getTitle() << endl;
                enrolledCount++;
            }
            ++enrolledIterator;  // Move to the next enrolled student.
        }

        // Iterate through the waitlisted students in the current course.
        while (waitlistIterator != courseArray[i].waitlist_end()) {
            // Check if the current student matches the provided ID and
name.
            if ((*waitlistIterator).getId() == studentId &&
(*waitlistIterator).getName() == studentName) {
                cout << "(W) " << courseArray[i].getCode() << "    " <<
courseArray[i].getTitle() << endl;
                waitlistCount++;
            }
            ++waitlistIterator;  // Move to the next waitlisted student.
        }
    }

    // Display the summary of the student's registrations and waitlist
status.
    cout << "\nYou registered in " << enrolledCount << " course(s) and
waitlisted in " << waitlistCount << " course(s)." << endl;
}



// Function to handle course registration for a student.
// Parameters:
//   courseArray: An array of Course objects representing available
courses.
//   courseCount: The number of courses in the courseArray.
void courseRegistration(Course* courseArray, int courseCount) {
    int studentId;
    string studentName;
    string courseCode;
    string courseTitle;

    cout << "Enter your ID: ";
    cin >> studentId;
    cout << "Enter your name: ";
    cin.ignore();
    getline(cin, studentName);
    cout << "Enter course code: ";
    cin >> courseCode;
    cout << "Enter course title: ";
    cin.ignore();
    getline(cin, courseTitle);
    bool registrationSucceeded = false;  // Flag to track if the
registration was successful.

    for (int i = 0; i < courseCount; i++) {
        // Check if the provided course code and title match a course in
the courseArray.
        if (courseArray[i].getCode() == courseCode &&
courseArray[i].getTitle() == courseTitle) {
            if (courseArray[i].getEnrolled() < 10) {
                // Enroll the student in the course if there is space in
the enrolled list.
                Student student(studentId, studentName);
                courseArray[i].addStudentToEnrolled(student);
                cout << "Registration succeeded!" << endl;
                registrationSucceeded = true;
                break;
            } else {
                // Waitlist the student for the course if the enrolled list
is full.
                Student student(studentId, studentName);
                courseArray[i].addStudentToWaitlist(student);
                cout << "You are on the waitlist for " << courseCode <<
endl;
                registrationSucceeded = true;
                break;
            }
        }
    }

    if (!registrationSucceeded) {
        cout << "Course not found or registration is full." << endl;
    }
}


// Function to handle course cancellation
void courseCancellation(Course* courseArray, int courseCount) {
    int studentId;
    string studentName;
    string courseCode;
    string courseTitle;

    cout << "Enter your ID: ";
    cin >> studentId;
    cout << "Enter your name: ";
    cin.ignore();
    getline(cin, studentName);
    cout << "Enter course code: ";
    cin >> courseCode;
    cout << "Enter course title: ";
    cin.ignore();
    getline(cin, courseTitle);

    bool removedFromEnrolled = false;
    bool removedFromWaitlist = false;

    // Loop through the available courses to check if the course matches
the input code and title
    for (int i = 0; i < courseCount; i++) {
        if (courseArray[i].getCode() == courseCode &&
courseArray[i].getTitle() == courseTitle) {
            // Check if the student is in the enrolled list of the course
            node* enrolledList = courseArray[i].getEnrolledList();
            node* previousEnrolled = nullptr;

            while (enrolledList != nullptr) {
                Student student = enrolledList->getData();
                if (student.getId() == studentId && student.getName() ==
studentName) {
                    // Remove the student from the enrolled list
                    courseArray[i].removeStudentFromEnrolled(student);
                    removedFromEnrolled = true;
                    cout << "'" << courseTitle << "' is removed from your
course list." << endl;
                    break;
                }
                previousEnrolled = enrolledList;
                enrolledList = enrolledList->getLink();
            }

            // If not found in the enrolled list, check the waitlist
            if (!removedFromEnrolled) {
                node* waitlist = courseArray[i].getWaitlistList();
                node* previousWaitlist = nullptr;

                while (waitlist != nullptr) {
                    Student student = waitlist->getData();
                    if (student.getId() == studentId && student.getName()
== studentName) {
                        // Remove the student from the waitlist
                        courseArray[i].removeStudentFromWaitlist(student);
                        removedFromWaitlist = true;
                        cout << "You have been removed from the waitlist of
'" << courseTitle << "'." << endl;
                        break;
                    }
                    previousWaitlist = waitlist;
                    waitlist = waitlist->getLink();
                }
            }

            // Once the student is found and removed, there's no need to
continue searching other courses
            if (removedFromEnrolled || removedFromWaitlist) {
                break;
            }
        }
    }

    // If the student wasn't removed from either the enrolled list or
waitlist, display a message
    if (!removedFromEnrolled && !removedFromWaitlist) {
        cout << "Course not found in your course list or you are not on the
waitlist." << endl;
    }
}


/// Function to print the enrollment list of courses, including students on
the waitlist.
// Parameters:
//   courseArray: An array of Course objects representing available
courses.
//   courseCount: The number of courses in the courseArray.
void printEnrollmentList(const Course* courseArray, int courseCount) {
    // Iterate through the courseArray to print enrollment details for each
course.
    for (int i = 0; i < courseCount; i++) {
        int enrolledCount = 0;     // Counter for the number of enrolled
```

```
students.
        int waitlistCount = 0;     // Counter for the number of students on
the waitlist.

        // Count the number of students in the enrolled list of the course.
        node* enrolledList = courseArray[i].getEnrolledList();
        while (enrolledList != nullptr) {
            enrolledCount++;
            enrolledList = enrolledList->getLink();
        }

        // Count the number of students on the waitlist for the course.
        node* waitlist = courseArray[i].getWaitlistList();
        while (waitlist != nullptr) {
            waitlistCount++;
            waitlist = waitlist->getLink();
        }

        // Print course information including enrollment and waitlist
counts.
        cout << "[ " << courseArray[i].getCode() << " " <<
courseArray[i].getTitle() << " (" << enrolledCount << ") ]" << endl;
        cout << "-------------------------------------" << endl;

        // Print the list of enrolled students for the course.
        enrolledList = courseArray[i].getEnrolledList();
        while (enrolledList != nullptr) {
            Student student = enrolledList->getData();
            cout << "    " << student.getId() << " " << student.getName() <<
endl;
            enrolledList = enrolledList->getLink();
        }

        if (waitlistCount > 0) {
            cout << "< Waitlist (" << waitlistCount << ") >" << endl;

            // Print the list of students on the waitlist for the course.
            waitlist = courseArray[i].getWaitlistList();
            while (waitlist != nullptr) {
                Student student = waitlist->getData();
                cout << "    " << student.getId() << " " <<
student.getName() << endl;
                waitlist = waitlist->getLink();
            }
        }

        cout << endl;  // Separate each course's information with an empty
line.
    }
}


int main() {
    const int MAX_COURSES = 11;
    Course courseArray[MAX_COURSES];
    int courseCount = 0;

    //file initialization
    string fileCourse;
    string fileEnroll;

    //user imputs file names
    cout << "Enter course filename : ";
    cin >> fileCourse;
    cout << "Enter enrollment filename : ";
    cin >> fileEnroll;

    //use functions to see if files open and add students to the arrays
    loadCourses(fileCourse, courseArray, courseCount);
    loadEnrollments(fileEnroll, courseArray, courseCount);

    int choice;

    //will do the fuction that is selected by the user imput
    do {
        choice = displayMenu();
        switch (choice) {
            case 1:
                viewRegistration(courseArray, courseCount);
                break;
            case 2:
                courseRegistration(courseArray, courseCount);
                break;
            case 3:
                courseCancellation(courseArray, courseCount);
                break;
            case 4:
                printEnrollmentList(courseArray, courseCount);
                break;
            case 5:
                cout << "Goodbye!" << endl;
                break;
            default:
                cout << "Invalid choice. Please try again." << endl;
        }
    } while (choice != 5);

    // Clean up dynamic memory
    // Loop through the courseArray to clean up dynamically allocated
memory.
for (int i = 0; i < courseCount; i++) {
    // Clean up memory for the enrolled students' list.
    node* enrolledList = courseArray[i].getEnrolledList();
    while (enrolledList != nullptr) {
        node* temp = enrolledList;  // Store the current node in a
temporary pointer.
        enrolledList = enrolledList->getLink();  // Move to the next node.
        delete temp;  // Delete the temporary node to free memory.
    }

    // Clean up memory for the waitlisted students' list.
    node* waitlist = courseArray[i].getWaitlistList();
    while (waitlist != nullptr) {
        node* temp = waitlist;  // Store the current node in a temporary
pointer.
        waitlist = waitlist->getLink();  // Move to the next node.
        delete temp;  // Delete the temporary node to free memory.
    }
}


    return 0;
}
```