**Project Description:**

\*Not in Project write up

\*In README file

**Main:**

#include <iostream> // cout, endl

#include <string> // std::string

#include "course.h" // Course class

#include "student.h" // Student class

#include "StudentDB.h" // StudentDB linked list database

void addStudent(StudentDB&); // Wrapper for adding a student

void updateStudent(StudentDB&); // Wrapper for updating a student

void removeStudent(StudentDB&); // Wrapper for removing a student

void addCourse(StudentDB&); // Wrapper for adding a course

void removeCourse(StudentDB&); // Wrapper for removing a course

void updateCourse(StudentDB&); // Wrapper for updating a course

void printDatabase(StudentDB&); // Wrapper for printing a database

int main() {

StudentDB list;

bool done = false; // loop flag

int choice; // input variable for user

puts("\t\tWelcome to the Student Database!\n");

std::cout << "\*Every list starts out empty and database is not saved when program commpletes." << std::endl; // Special directions for correct use of program

std::cout << "\*Every name and course that is searched for must be entered exactly as it is shown in the database in order to ensure correct operations" << std::endl;

do {

std::cout << "Command List:\n"; // Instructions posted after every loop

std::cout << "1. Add Student\n";

std::cout << "2. Update Student\n";

std::cout << "3. Remove Student\n";

std::cout << "4. Add Course\n";

std::cout << "5. Remove Course\n";

std::cout << "6. Update Course\n";

std::cout << "7. Print Database\n";

std::cout << "8. Quit\n";

std::cout << "\nPlease enter a command (1 - 8): ";

std::cin >> choice;

if (choice < 1 || choice > 8)

continue;

switch (choice) { // Commands call each specific function

case 1:

addStudent(list);

break;

case 2:

updateStudent(list);

break;

case 3:

removeStudent(list);

break;

case 4:

addCourse(list);

break;

case 5:

removeCourse(list);

break;

case 6:

updateCourse(list);

break;

case 7:

printDatabase(list);

break;

case 8:

done = true;

break;

}

} while(!done);

return 0;

}

/\*

\* addStudent Function:

\* Takes a list reference and adds a student to the list after the needed information

\* is received in order to create the student

\*/

void addStudent(StudentDB &l) {

std::string name, dob, major;

std::cout << "Enter a student name (LastName,FirstName) of (FirstName): ";

std::cin >> name;

std::cout << "\nEnter the student's date of birth (Format ##/##/####): ";

std::cin >> dob;

std::cout << "\nEnter the students major (Must be one word or abbreviation): ";

std::cin >> major;

Student s1(name, dob, major);

l.addStudent(s1);

}

/\*

\* updateStudent Function:

\* Takes a list reference and updates a specified student by the user

\*/

void updateStudent(StudentDB &l) {

std::cout << "\*\*Enter exactly as displayed in database! " << std::endl;

std::string newName, newDOB, newMajor, srchName;

std::cout << "Enter the new student name: ";

std::cin >> newName;

std::cout << "\nEnter the new student's date of birth: ";

std::cin >> newDOB;

std::cout << "\nEnter the new student's major: ";

std::cin >> newMajor;

std::cout << "\nWhat is the name of the student to update in the database?";

std::cin >> srchName;

Student s1(newName, newDOB, newMajor);

l.updateStudent(s1, srchName);

}

/\*

\* removeStudent Function:

\* Takes a list reference and removes a specified student by the user

\*/

void removeStudent(StudentDB &l) {

std::cout << "\*\*Enter exactly as displayed in database! " << std::endl;

std::string srchName;

std::cout << "Enter the name of the student you want to remove from the list: ";

std::cin >> srchName;

l.removeStudent(srchName);

}

/\*

\* addCourse Function:

\* Takes a list reference and adds a course to a specified student by the user after

\* the info for creating is course is also specified

\*/

void addCourse(StudentDB &l) {

std::string name, depart, sem, srchName;

char grade;

std::cout << "Enter the name of the course (NameOfCourse): ";

std::cin >> name;

std::cout << "\nEnter the department of the course (One word or abbreviation): ";

std::cin >> depart;

std::cout << "\nEnter the semester of the course (fa/sp ## year ex. fa19): ";

std::cin >> sem;

do {

std::cout << "\nEnter the grade earned for the course (capital character A-F): ";

std::cin >> grade;

} while ((grade < 'A') || (grade > 'F'));

std::cout << "\n\*\*Enter exactly as displayed in database! ";

std::cout << "\nEnter the name of the student for the course to be added to: ";

std::cin >> srchName;

Course c1(name, depart, sem, grade);

l.addCourse(c1, srchName);

}

/\*

\* removeCourse Function:

\* Takes a list reference and removes a specified course from a specified student from the

\* user

\*/

void removeCourse(StudentDB &l) {

std::cout << "\*\*Enter exactly as displayed in database! " << std::endl;

std::string srchName, srchCourse;

std::cout << "Enter the name of the student whose course you want to remove: ";

std::cin >> srchName;

std::cout << "\nEnter the name of the course to be removed: ";

std::cin >> srchCourse;

l.removeCourse(srchName, srchCourse);

}

/\* updateCourse Function:

\* Takes a list reference and updates a specified course to a specified student after the new

\* course data is entered by the user

\*/

void updateCourse(StudentDB &l) {

std::cout << "\*\*Enter exactly as displayed in database! " << std::endl;

std::string newName, newDepart, newSem, srchName, srchCourse;

char newGrade;

std::cout << "Enter the new name of the course: ";

std::cin >> newName;

std::cout << "\nEnter the new department of the course: ";

std::cin >> newDepart;

std::cout << "\nEnter the new semester of the course: ";

std::cin >> newSem;

std::cout << "\nEnter the new grade of the course: ";

std::cin >> newGrade;

std::cout << "\nEnter the name of the student whose course will be updated: ";

std::cin >> srchName;

std::cout << "\nEnter the course of the student that will be updated: ";

std::cin >> srchCourse;

Course c1(newName, newDepart, newSem, newGrade);

l.updateCourse(srchName, srchCourse, c1);

}

/\*

\* printDatabase Function:

\* Takes a list reference and prints the database to the screen

\*/

void printDatabase(StudentDB &l) {

std::cout << "\tDATABASE:\n";

l.printDatabase();

}

**Sample Output:**

\*Sample output submitted in document online

**Course.h:**

#ifndef \_COURSE\_H\_

#define \_COURSE\_H\_

#include <iostream>

#include <string>

class Course {

private:

std::string name;

std::string department;

std::string semester;

char grade;

public:

Course(); // Default ctor

Course(std::string, std::string, std::string, char); // Overloaded ctor

std::string getName(); // returns name

std::string getDepartment(); // returns department

std::string getSemester(); // returns semester

char getGrade(); // returns grade

void setName(std::string); // sets name

void setDepartment(std::string); // sets department

void setSemester(std::string); // sets semester

void setGrade(char); // sets grade

void printCourseInfo(); // prints all course info

};

#endif

**Course.cpp:**

#include "course.h"

/\*

\* Default Constructor:

\* This will initialize all private members to default values, as well as

\* initialize the grade to an A in case of no members sent in to initialize

\* them otherwise

\*/

Course::Course() {

name = "";

department = "";

semester = "";

grade = 'A';

}

/\*

\* Overloaded Constructor (4 arguments):

\* This will initialize all members to what is passed through by the user

\*/

Course::Course(std::string n, std::string d, std::string s, char g) {

name = n;

department = d;

semester = s;

grade = g;

}

/\*

\* getName Function:

\* Returns a string with the name of the course

\*/

std::string Course::getName() {

return name;

}

/\*

\* getDepartment Function:

\* Returns a string with the name of the department

\*/

std::string Course::getDepartment() {

return department;

}

/\*

\* getSemester Function:

\* Returns a string with the name of the semester

\*/

std::string Course::getSemester() {

return semester;

}

/\*

\* getGrade Function:

\* Returns a character that is the grade of the course

\*/

char Course::getGrade() {

return grade;

}

/\*

\* setName Function:

\* Takes a string parameter and sets the name to it

\*/

void Course::setName(std::string n) {

name = n;

}

/\*

\* setDepartment Function:

\* Takes a string parameter and sets the department to it

\*/

void Course::setDepartment(std::string d) {

department = d;

}

/\*

\* setSemester Function:

\* Takes a string parameter and sets the semester to it

\*/

void Course::setSemester(std::string s) {

semester = s;

}

/\*

\* setGrade Function:

\* Takes a char parameter and sets the grade to it

\*/

void Course::setGrade(char g) {

grade = g;

}

/\*

\* printCourseInfo Function:

\* Prints out all of the course info

\*/

void Course::printCourseInfo() {

std::cout << "==================================" << std::endl;

std::cout << "Course Name: " << name << std::endl;

std::cout << "Department: " << department << std::endl;

std::cout << "Semester: " << semester << std::endl;

std::cout << "Grade Received: " << grade << std::endl;

std::cout << "==================================\n" << std::endl;

}

**Student.h:**

#ifndef \_STUDENT\_H\_

#define \_STUDENT\_H\_

#include <iostream>

#include <string>

class Student {

private:

std::string name;

std::string dob;

std::string major;

public:

Student(); // Default ctor

Student(std::string, std::string, std::string); // Overloaded ctor

std::string getName(); // returns name

std::string getDOB(); // returns dob

std::string getMajor(); // returns major

void setName(std::string); // sets name

void setDOB(std::string); // sets dob

void setMajor(std::string); // sets major

void printStudentInfo(); // prints out all Student info

};

#endif

**Student.cpp:**

#include "student.h"

/\*

\* Default Constructor:

\* Initializes all values to default values in case nothing is input

\*/

Student::Student() {

name = "";

dob = "";

major = "";

}

/\*

\* Overloaded Constructor:

\* Initializes all values to the strings that are passed through

\*/

Student::Student(std::string n, std::string d, std::string m) {

name = n;

dob = d;

major = m;

}

/\*

\* getName Function:

\* Returns a string that is the name of the student

\*/

std::string Student::getName() {

return name;

}

/\*

\* getDOB Function:

\* Returns a string that is the date of birth of the student

\*/

std::string Student::getDOB() {

return dob;

}

/\*

\* getMajor Function:

\* Returns a string that is the major of the student

\*/

std::string Student::getMajor() {

return major;

}

/\*

\* setName Function:

\* Takes a string as a parameter and sets name to it

\*/

void Student::setName(std::string n) {

name = n;

}

/\*

\* setDOB Function:

\* Takes a string as a parameter and sets dob to it

\*/

void Student::setDOB(std::string d) {

dob = d;

}

/\*

\* setMajor Function:

\* Takes a string as a parameter and sets major to it

\*/

void Student::setMajor(std::string m) {

major = m;

}

/\*

\* printStudentInfo Function:

\* Prints out all of the private members of the current Student

\*/

void Student::printStudentInfo() {

std::cout << "==========================" << std::endl;

std::cout << "Name: " << name << std::endl;

std::cout << "Date of Birth: " << dob << std::endl;

std::cout << "Major: " << major << std::endl;

std::cout << "==========================" << std::endl;

}

**StudentDB.h:**

#ifndef \_STUDENTDB\_H\_

#define \_STUDENTDB\_H\_

#include <string>

#include <iostream>

#include "course.h"

#include "student.h"

class StudentDB {

private:

struct CourseNode {

Course c;

CourseNode\* cnext;

};

struct StudentNode {

Student s;

StudentNode\* snext;

CourseNode\* chead;

};

StudentNode\* head;

StudentNode \*findStudent(std::string);

CourseNode \*findCourse(std::string, std::string);

public:

StudentDB(); // Default ctor

~StudentDB(); // Destructor

StudentDB(const StudentDB&); // Copy ctor

StudentDB operator=(const StudentDB&); // Overloaded assignment

void addStudent(Student); // adds a student to list

void updateStudent(Student, std::string); // updates a student to new student passed

void removeStudent(std::string); // removes a student

void addCourse(Course, std::string); // pass name of student and new Course obj

void removeCourse(std::string, std::string); // removes Course from student passed

void updateCourse(std::string, std::string, Course); // updates course at student and course passed through

void printDatabase(); // prints all students and all their corresponding courses

};

#endif

**StudentDB.cpp:**

#include "StudentDB.h"

/\*

\* findStudent Function:

\* Takes a string that is the name of the student to be searched for, as well as a pointer to the

\* head of the list, and returns a StudentNode pointer to the student that was being searched for

\*/

StudentDB::StudentNode\* StudentDB::findStudent(std::string search) {

StudentNode\* cursor = head;

if (head == nullptr) {

std::cout << "The list is empty and no student can be searched for" << std::endl;

return nullptr;

}

while (cursor) {

if (search.compare(cursor->s.getName()) == 0) {

return cursor;

}

cursor = cursor->snext;

}

return nullptr;

}

/\*

\* findCourse Function:

\* Takes a string that is the name of the course that is to be searched for and returns a CourseNode

\* pointer to the Course that was being searched for

\*/

StudentDB::CourseNode\* StudentDB::findCourse(std::string stud, std::string search) {

StudentNode\* currStud = findStudent(stud);

if (currStud == nullptr) {

std::cout << "No student was found! Please enter the name exactly as displayed." << std::endl;

return nullptr;

}

if (currStud->chead == nullptr) {

std::cout << "No courses are assigned to the current student" << std::endl;

return nullptr;

}

CourseNode\* cursor = currStud->chead;

while (cursor) {

if (search.compare(cursor->c.getName()) == 0) {

return cursor;

}

cursor = cursor->cnext;

}

return nullptr;

}

/\* Copy Constructor:

\* This constructor copies the list over for a deep copy

\*/

StudentDB::StudentDB(const StudentDB &rhs) {

head = nullptr;

StudentNode\* cursor = rhs.head;

if (cursor == nullptr) {

return;

}

CourseNode\* down = cursor->chead;

while (cursor) {

addStudent(cursor->s);

while (down) {

addCourse(down->c, cursor->s.getName());

down = down->cnext;

}

cursor = cursor->snext;

if (cursor) {

down = cursor->chead;

}

}

}

/\* Overloaded Assignment:

\* Allows the ability to assign one list to another to copy it over to the other list

\*/

StudentDB StudentDB::operator=(const StudentDB &rhs) {

if (this == &rhs) {

return \*this;

}

// Delete current objects dynamic memory

StudentNode\* dCursor = head;

CourseNode\* dDown;

CourseNode\* follow;

while (dCursor) {

dCursor = dCursor->snext;

dDown = head->chead;

follow = dDown;

delete head;

while (dDown) {

dDown = dDown->cnext;

delete follow;

follow = dDown;

}

head = dCursor;

}

// Copy everything over

head = nullptr;

StudentNode\* cursor = rhs.head;

if (cursor == nullptr) {

StudentDB temp;

return temp;

}

CourseNode\* down = cursor->chead;

while (cursor) {

addStudent(cursor->s);

while (down) {

addCourse(down->c, cursor->s.getName());

down = down->cnext;

}

cursor = cursor->snext;

if (cursor) {

down = cursor->chead;

}

}

return \*this;

}

/\*

\* Default Constructor:

\* Initializes list to empty

\*/

StudentDB::StudentDB() {

head = nullptr;

}

/\*

\* Destructor:

\* Destroys all dynamically allocated memory

\*/

StudentDB::~StudentDB() {

StudentNode\* cursor = head;

CourseNode\* down;

CourseNode\* follow;

while (cursor) {

cursor = cursor->snext;

down = head->chead;

follow = down;

delete head;

while (down) {

down = down->cnext;

delete follow;

follow = down;

}

head = cursor;

}

}

/\*

\* addStudent Function:

\* Takes a Student object and appends a student node to the list

\*/

void StudentDB::addStudent(Student obj) {

StudentNode\* newNode = new StudentNode;

newNode->s = obj;

newNode->snext = nullptr;

newNode->chead = nullptr;

if (head == nullptr) {

head = newNode;

return;

}

StudentNode\* cursor = head;

while (cursor->snext) {

cursor = cursor->snext;

}

cursor->snext = newNode;

}

/\*

\* updateStudent Function:

\* Takes a Student object, and a string that will be the name of the student

\* that needs to be updated, the object passed will replace the student node with

\* the name that was passed through

\*/

void StudentDB::updateStudent(Student obj, std::string srchName) {

StudentNode\* currStud = findStudent(srchName);

if (currStud == nullptr) {

std::cout << "Name not found! Please enter search name exactly as displayed in database" << std::endl;

return;

}

currStud->s = obj;

}

/\*

\* removeStudent Function:

\* Takes a string that is the name of a student to be removed and then is removed

\* from the list

\*/

void StudentDB::removeStudent(std::string srchName) {

StudentNode\* currStud = findStudent(srchName);

if (currStud == nullptr) {

std::cout << "Name not found! Please enter the search name exactly as displayed in database" << std::endl;

return;

}

if (currStud == head) {

head = head->snext;

CourseNode\* down = currStud->chead;

while (down) {

down = down->cnext;

delete currStud->chead;

currStud->chead = down;

}

delete currStud;

return;

}

StudentNode\* cursor = head;

while (cursor->snext != currStud) {

cursor = cursor->snext;

}

if (currStud->snext == nullptr) {

cursor->snext = nullptr;

CourseNode\* down = currStud->chead;

while (down) {

down = down->cnext;

delete currStud->chead;

currStud->chead = down;

}

delete currStud;

return;

}

cursor->snext = currStud->snext;

CourseNode\* down = currStud->chead;

while (down) {

down = down->cnext;

delete currStud->chead;

currStud->chead = down;

}

delete currStud;

}

/\*

\* addCourse Function:

\* Takes a Course object and a string of the student that the course will be appended to

\*/

void StudentDB::addCourse(Course obj, std::string srchName) {

StudentNode\* currStud = findStudent(srchName);

if (currStud == nullptr) {

std::cout << "Name not found in list. (Name must be exactly as found in database)" << std::endl;

return;

}

CourseNode\* newNode = new CourseNode;

newNode->c = obj;

newNode->cnext = nullptr;

if (currStud->chead == nullptr) {

currStud->chead = newNode;

return;

}

CourseNode\* cursor = currStud->chead;

while (cursor->cnext) {

cursor = cursor->cnext;

}

cursor->cnext = newNode;

}

/\*

\* removeCourse Function:

\* Takes a string of the student that has the course to be removed as well as the course

\* to be removed and then will be removed from the current student in the list

\*/

void StudentDB::removeCourse(std::string srchStud, std::string srchCrse) {

StudentNode\* currStud = findStudent(srchStud);

if (currStud == nullptr) {

std::cout << "No student found in list! Please enter name exactly as they are displayed" << std::endl;

return;

}

CourseNode\* currCourse = findCourse(srchStud, srchCrse);

if (currCourse == nullptr) {

std::cout << "No course found in list! Please enter name exactly as they are displayed" << std::endl;

return;

}

if (currCourse == currStud->chead) {

currStud->chead = currStud->chead->cnext;

delete currCourse;

return;

}

CourseNode\* cursor = currStud->chead;

while (cursor->cnext != currCourse) {

cursor = cursor->cnext;

}

if (currCourse->cnext == nullptr) {

cursor->cnext = nullptr;

delete currCourse;

return;

}

cursor->cnext = currCourse->cnext;

delete currCourse;

}

/\*

\* updateCoures Function:

\* Takes a string of the student name and a string of a course name that belongs to the student,

\* and the Course object to update the course that was passed for the student

\*/

void StudentDB::updateCourse(std::string srchStud, std::string srchCrse, Course obj) {

CourseNode\* currCourse = findCourse(srchStud, srchCrse);

if (currCourse == nullptr) {

std::cout << "Course not found on current list! Please type in course name exactly as displayed in database." << std::endl;

return;

}

currCourse->c = obj;

}

/\*

\* printDatabase Function:

\* Prints all of the students and all their corresponding courses from the list

\*/

void StudentDB::printDatabase() {

if (head == nullptr) {

std::cout << "The list is empty, nothing to output" << std::endl;

return;

}

StudentNode\* cursor = head;

CourseNode\* cCursor = cursor->chead;

while (cursor) {

cursor->s.printStudentInfo();

while (cCursor) {

if (cCursor == cursor->chead) {

std::cout << "------------Courses------------\n";

}

cCursor->c.printCourseInfo();

cCursor = cCursor->cnext;

}

if (cursor->chead == nullptr) {

std::cout << "Currently no courses assigned" << std::endl;

}

cursor = cursor->snext;

if (cursor != nullptr) {

cCursor = cursor->chead;

}

}

}