// Austin Faulkner: May 24, 2020

// Specification file for the StudentData class

#ifndef STUDENTDATA\_H

#define STUDENTDATA\_H

#include <string>

#include <iostream>

#include <fstream>

// #include <exception>

class StudentData

{

public:

// HOW CAN I MAKE THESE ATTRIBUTES PRIVATE AND STILL SORT THE LL WITH METHOD

// SortLinkedList()?

std::string studentFirstName;

std::string studentMiddleName;

std::string studentLastName;

std::string studentId;

std::string studentMajor;

double gpa;

std::string residency;

std::string streetNumber;

std::string streetName;

std::string driveStreetCourt;

std::string apartmentNum;

std::string city;

std::string state;

int zipcode;

public:

// Constructors

StudentData()

{ studentFirstName = " ";

studentMiddleName = " ";

studentLastName = " ";

studentId = " ";

studentMajor = " ";

gpa = 0.0;

residency = " ";

streetNumber = " ";

streetName = " ";

driveStreetCourt = " ";

apartmentNum = " ";

city = " ";

state = " ";

zipcode = 0; }

StudentData(std::string f, std::string m, std::string l, std::string id,

std::string maj, double g, std::string r, std::string snum,

std::string sname, std::string dsc, std::string apartNum,

std::string c, std::string s, int z)

{ studentFirstName = f;

studentMiddleName = m;

studentLastName = l;

studentId = id;

studentMajor = maj;

gpa = g;

residency = r;

streetNumber = snum;

streetName = sname;

driveStreetCourt = dsc;

apartmentNum = apartNum;

city = c;

state = s;

zipcode = z; }

// Setters & Getters

void setFirstName(std::string f)

{ studentFirstName = f; }

std::string getFirstName() const

{ return studentFirstName; }

void setMiddleName(std::string m)

{ studentMiddleName = m; }

std::string getMiddleName() const

{ return studentMiddleName; }

void setLastName(std::string l)

{ studentLastName = l; }

std::string getLastName() const

{ return studentLastName; }

void setID(std::string id)

{ studentId = id; }

std::string getID() const

{ return studentId; }

void setMajor(std::string maj)

{ studentMajor = maj; }

std::string getMajor() const

{ return studentMajor; }

void setGPA(double g)

{ gpa = g; }

double getGPA() const

{ return gpa; }

void setResidency(std::string r)

{ residency = r; }

std::string getResidency() const

{ return residency; }

void setStreetNum(std::string snum)

{ streetNumber = snum; }

std::string getStreetNum() const

{ return streetNumber; }

void setStreetName(std::string sname)

{ streetName = sname; }

std::string getStreetName() const

{ return streetName; }

void setDSC(std::string dsc)

{ driveStreetCourt = dsc; }

std::string getDSC() const

{ return driveStreetCourt; }

void setApartNum(std::string apartNum)

{ apartmentNum = apartNum; }

std::string getApartNum() const

{ return apartmentNum; }

void setCity(std::string c)

{ city = c; }

std::string getCity() const

{ return city; }

void setState(std::string s)

{ state = s; }

std::string getState() const

{ return state; }

void setZipcode(int z)

{ zipcode = z; }

int getZipcode() const

{ return zipcode; }

// Friends

friend std::ostream &operator << (std::ostream& outs,

const StudentData& database)

{

outs << std::endl << std::endl

<< "Name: "

<< database.getFirstName()

<< " "

<< database.getMiddleName()

<< " "

<< database.getLastName() << std::endl

<< "Student ID: "

<< database.getID() << std::endl

<< "Major: "

<< database.getMajor() << std::endl

<< "Grade Point Average: "

<< database.getGPA() << std::endl

<< "Residency Status: "

<< database.getResidency() << std::endl

<< "Address: " << std::endl

<< "\t\t" << database.getStreetNum() << " "

<< database.getStreetName()

<< " " << database.getDSC() << std::endl

<< "\t\t" << "P.O. Box / Apt. " << database.getApartNum()

<< std::endl

<< "\t\t" << database.getCity() << ", " << database.getState() << " "

<< database.getZipcode() << std::endl;

return outs;

}

// friend std::ifstream &operator >> (std::ifstream& fin,

// StudentData& database)

// {

// fin >> database.studentFirstName

// >> database.studentMiddleName

// >> database.studentLastName

// >> database.studentId

// >> database.studentMajor

// >> database.gpa

// >> database.residency

// >> database.streetNumber

// >> database.streetName

// >> database.driveStreetCourt

// >> database.apartmentNum

// >> database.city

// >> database.state

// >> database.zipcode;

//

// ins.close();

//

// return ins;

// }

friend std::ofstream &operator << (std::ofstream& fout,

const StudentData& database)

{

fout.open("student\_output.txt", std::ofstream::out | std::ofstream::app);

fout << std::endl << std::endl

<< "Name: "

<< database.getFirstName()

<< " "

<< database.getMiddleName()

<< " "

<< database.getLastName() << std::endl

<< "Student ID: "

<< database.getID() << std::endl

<< "Major: "

<< database.getMajor() << std::endl

<< "Grade Point Average: "

<< database.getGPA() << std::endl

<< "Residency Status: "

<< database.getResidency() << std::endl

<< "Address: " << std::endl

<< "\t\t\t" << database.getStreetNum() << " "

<< database.getStreetName()

<< " " << database.getDSC() << std::endl

<< "\t\t\t" << "P.O. Box / Apt. " << database.getApartNum()

<< std::endl

<< "\t\t\t" << database.getCity() << ", " << database.getState()

<< " "

<< database.getZipcode() << std::endl << std::endl;

fout.close();

return fout;

}

friend std::istream &operator >> (std::istream& ins,

StudentData& database)

// Needs to be fixed.

{

std::string studentFirstName;

ins >> studentFirstName;

database.setFirstName(studentFirstName);

std::string studentMiddleName;

ins >> studentMiddleName;

database.setMiddleName(studentMiddleName);

std::string studentLastName;

ins >> studentLastName;

database.setLastName(studentLastName);

std::string studentId;

ins >> studentId;

database.setID(studentId);

std::string studentMajor;

ins >> studentMajor;

database.setMajor(studentMajor);

double gpa;

ins >> gpa;

database.setGPA(gpa);

std::string residency;

ins >> residency;

database.setResidency(residency);

std::string streetNumber;

ins >> streetNumber;

database.setStreetNum(streetNumber);

std::string streetName;

ins >> streetName;

database.setStreetName(streetName);

std::string driveStreetCourt;

ins >> driveStreetCourt;

database.setDSC(driveStreetCourt);

std::string apartmentNum;

ins >> apartmentNum;

database.setApartNum(apartmentNum);

std::string city;

ins >> city;

database.setCity(city);

std::string state;

ins >> state;

database.setState(state);

int zipcode;

ins >> zipcode;

database.setZipcode(zipcode);

// ins >> database.studentFirstName

// >> database.studentMiddleName

// >> database.studentLastName

// >> database.studentId

// >> database.studentMajor

// >> database.gpa

// >> database.residency

// >> database.streetNumber

// >> database.streetName

// >> database.driveStreetCourt

// >> database.apartmentNum

// >> database.city

// >> database.state

// >> database.zipcode;

return ins;

}

// GOOD OPPORTUNITY FOR A TRY-CATCH BLOCK: WHAT exception(s)?

bool operator == (const StudentData& rhs) const

{

return studentFirstName == rhs.studentFirstName &&

studentMiddleName == rhs.studentMiddleName &&

studentLastName == rhs.studentLastName &&

studentId == rhs.studentId &&

studentMajor == rhs.studentMajor &&

gpa == rhs.gpa &&

residency == rhs.residency &&

streetNumber == rhs.streetNumber &&

streetName == rhs.streetName &&

driveStreetCourt == rhs.driveStreetCourt &&

apartmentNum == rhs.apartmentNum &&

city == rhs.city &&

state == rhs.state &&

zipcode == rhs.zipcode;

}

// GOOD OPPORTUNITY FOR A TRY-CATCH BLOCK: WHAT exception(s)?

bool operator != (const StudentData& rhs) const

{

return studentFirstName == rhs.studentFirstName &&

studentMiddleName == rhs.studentMiddleName &&

studentLastName == rhs.studentLastName &&

studentId == rhs.studentId &&

studentMajor == rhs.studentMajor &&

gpa == rhs.gpa &&

residency == rhs.residency &&

streetNumber == rhs.streetNumber &&

streetName == rhs.streetName &&

driveStreetCourt == rhs.driveStreetCourt &&

apartmentNum == rhs.apartmentNum &&

city == rhs.city &&

state == rhs.state &&

zipcode == rhs.zipcode;

}

};

#endif