Michael Hernandez

BCS 230 Capstone Project Final Report

Professor Ilknur Aydin

|  |  |  |
| --- | --- | --- |
| **FINAL PROJECT** | | |
|  | **Student Self-Assessment** | **Professor Aydin’s Comments** |
| **Parts of the Project (47 pts)** | | |
| Entire Project Compiles  without any error or warning  (-10 pts if project does not compile) | Yes |  |
| Welcome Screen works  (1  pts) | Yes |  |
| Main menu loop works  (2 pts) | Yes |  |
| One Sub Menu loop work for each list  (2  pts) | Yes |  |
| ‘List all’ option for all three lists work and each list is sorted based on ID  (using Bubble sort algorithm) (5 pts)  Extra Credit: sorting can be done by ID or name depending on what criteria the user chooses  (+5 pts) | Yes, bubblesort algorithm used, only based on ID |  |
| ‘Add one’ option for all three lists work  (3 pts) | Yes |  |
| ‘Search one’ option for all three lists work  (3 pts)  Extra Credit: ‘Search one’ menu options in each sub menu is completed using Binary Search (+5 pts) | Yes, binary search not included |  |
| ‘Edit one’ option for all three lists work (extra credit, +5 pts) | Yes |  |
| ‘Delete one’ option for all three lists work (extra credit, +5 pts) | Not included |  |
| ‘Return to main menu’ option works from each 3 sub menu  (2  pts) | Yes |  |
| Department List  Each department has: Department Name, Department ID  (1 pts) | Yes |  |
| At the start of your program: The department list  is correctly read from an input file formatted as CSV into the corresponding list in the program  (2 pts) | Yes problem with input was fixed |  |
| At the end of your program: The *updated* department list is correctly saved to an output text file in CSV format.  (2 pts) | Yes |  |
| The hospital will have three types of staff: Doctors, Nurses, Administrative Personnel  (2 pts) | Yes |  |
| Each staff member has following information:  First Name, Last Name, Staff ID, Department, Personal info, Contact info  where   * Staff ID: will be a unique number per staff member * Department: must be one of the department ID in the system as each staff is a member of a department. * Personal info will include: birth date, gender * Contact info will include:   phone info, email info * Phone info: can be a single phone number or a list of phone numbers (extra credit. +5 pts) * Email info: can be a single email address or a list of email addresses (extra credit, +5 pts)   (6 pts) | Yes, multiple phone numbers and email addresses can be stored |  |
| At the start of your program: The staff list  is correctly read from an input file formatted as CSV into the corresponding list in the program  (2 pts) | Yes, problem with input was fixed |  |
| At the end of your program: The *updated* staff list is correctly saved to an output text file in CSV format.  (2pts) | Yes |  |
| Patient List  Each patient has following information:  First Name, Last Name, Patient ID, Doctor, Personal info, Contact info  where   * Patient ID: is a unique number per patient * Doctor: must be ID of one of the of the doctors in the system, since each patient is assigned one doctor * Personal info will include: birth date, gender * Contact info will include:   home address info, phone info, email info * home address info will include: no, street, city, state, zip * phone info: can be a single phone number or a list of phone numbers (extra credit, +5 pts) * email info: can be a single email address or a list of email addresses (extra credit, +5 pts)   (6 pts) | Yes |  |
| At the start of your program: The patient list  is correctly read from an input file formatted as CSV into the corresponding list in the program  (3 pts) | Yes problem with input was fixed |  |
| At the end of your program: The *updated* patient list is correctly saved to an output text file in CSV format.  (3 pts) | Yes |  |
| **Requirements of the Final Project (47 points)** | | |
| Appropriate use of Functions  in the entire project  (8  pts) | Yes all functions serve a purpose |  |
| Appropriate use of Comments  (2  pts) | All functions and variables have comments to explain them |  |
| Appropriate use of Variable Names  (2  pts) | Yes |  |
| Arrays used correctly  (4  pts) | Yes |  |
| At least one struct is used  (3  pts) | Yes several structs used |  |
| At least three different C++ classes designed and implemented correctly  (15 pts) | Yes |  |
| At least one inheritance  (4  pts) | Yes both Patient and Staff inherit from Person Class |  |
| At least one composition  (4 pts) | Yes Staff has a composition of type Department |  |
| At least one dynamic array used correctly (making sure new and delete keywords are used appropriately)  (5 pts) | Yes, StaffContactInfo has two dynamic arrays, email and phoneNo that has a constructor, copy constructor, destructor and overloaded assignment operator |  |
| **Submission (6 pts)** | | |
| Project report is correctly prepared and includes all the required sections  (4  pts) | Yes all parts are included |  |
| Correct Files are submitted and in a way as requested in the project description  (2 pts) | Yes |  |

Functions.h

//Author: Michael Hernandez

//BCS 230 Capstone Project: Midterm draft

//Date last modified: May 7, 2016

#pragma once

using namespace std;

struct BirthDate {

int day;//stores birth day

int month;// stores birth month

int year;//stores birth year

};//struct to store birth dates

class Person {

private:

char gender;//stores gender as a character

BirthDate birthD;//stores birthDate which is a struct of three integers listed above

string firstName;//stores first name of a person

string lastName;//stores last name of a person

public:

void setGender(char);// sets gender variable

void setFirstName(string);//sets firstName variable

void setLastName(string);//sets lastName variable

void setBirthDay(int);// sets birthD.day variable

void setBirthMonth(int);//sets birthD.month variable

void setBirthYear(int);//sets birthD.year variable

char getGender() const;//returns gender variable

string getFirstName() const;//returns firstName variable

string getLastName() const;//returns lastName variable

int getBirthDay() const;//returns birthD.day variable

int getBirthMonth() const;//returns birthD.month variable

int getBirthYear() const;//returns birthD.year variable

Person(char = 'M', int = 1, int = 1, int = 1990, string = "first", string = "last");//default constructor with default parameters

};//class to store and manipulate personal information

class StaffContactInfo {

private:

int phoneNoAmount;//stores the amount of phone numbers to be stored for a staff member

int emailAmount;//stores the amount of e-mail addresses to be stored for a staff member

public:

string \*email;//stores e-mail address as a string for a staff member

string \*phoneNo;//stores phone number as a string for a staff member

void setNumOfEmail(int);//sets the value of emailAmount

void setNumofPhones(int);//sets the value of phoneNoAmount

void setANumber(string, int);//sets a number in the phoneNo dynamic array

void setAnEmail(string, int);//sets a number in the email dynamic array

int getPhoneNoAmount() const;//returns phoneNoAmount

int getEmailAmount() const;//returns emailAmount

const StaffContactInfo& operator= (const StaffContactInfo&);//overloads assignment operator

StaffContactInfo(const StaffContactInfo&);//copy constructor

StaffContactInfo(int = 1, int = 1);//default constructor with default parameters

~StaffContactInfo();//destructor

};//struct to store contact information about staff members

struct HomeAddressInfo {

int addNo;//stores address number as an integer

string streetName;// stores street name as a string

string cityName;//stores city name as a string

string stateName;//stores state name as a string

int zipCode;//stores zip code as an integer

};//struct to store address info

struct PatientContactInfo {

string email;//stores e-mail address of a patient

string phoneNo;//stores phone number of a patient

HomeAddressInfo patientAddress;// stores address information for a patient

};//struct to store contact information about patients

class Department {

private:

string deptName;//stores name of a department

int ID;//stores ID of a department

public:

void setDeptName(string);//sets deptName

void setDeptID(int);// sets ID

string getDeptName() const;//returns deptName

int getDeptID() const;//returns ID

void readFromFile(ifstream&);//reads data about a department from a file

void printToFile(ofstream&);//prints data about a department to an output file

void printDepartment() const;//prints data about a department to the screen

Department();//default constructor for Department

};

class Staff

:public Person

{

private:

char staffType;//stores the type of staff member as a character (D, N, or A)

int staffID;//stores the ID number for a staff member

public:

Department staffDept;//composition

StaffContactInfo staffContact;//struct to store contact information of a staff member

void setStaffID(int);//sets staffID

void setStaffType(char);//sets staffType

void readFromFile(ifstream&);//reads info about a staff member from a file

void printToFile(ofstream&);//prints info about a staff member to an output file

int getStaffID() const;//returns staffID

char getStaffType() const; //returns staffType

void printStaff() const;//prints information about a staff member to the screen

Staff(int = 0, int = 0, char = 'M', int = 1, int = 1, int = 1990, int = 1, int = 1, string = "", string = "", char = 'A');//default constructor with default parameters for Staff

};

class Patient

:public Person

{

private:

int patientID;//stores patient ID

int attendingStaffID;//stores the ID of the attending physician for a Patient

PatientContactInfo patientContact;//struct that stores contact information for a patient

public:

void setID(int);//sets patientID

void setAttendingID(int);//sets attendingStaffID

int getAttendingID() const;//returns attendingStaffID

void setAddress(int, string, string, string, int);//sets the address fields of patientContact

void setEmail(string);//sets the e-mail of patientContact

void setPhoneNum(string);//set the phone number of patientContact

void readFromFile(ifstream&);//reads info about a patient from a file

void printToFile(ofstream&);//outputs info about a patient onto an output file in CSV format

void setBirthday(int, int, int);//set a patient's birthday (for editing a patient's info)

int getPatientID() const;//returns patientID

void setPatientContactInfo(PatientContactInfo);//sets all variables in a patient's patientContact

void printPatientInfo() const;//prints information about a patient

Patient(int = 0, int = 0, char = 'M', int = 1, int = 1, int = 1990, string = "", string = "", string = "", string = "",

int = 001, string = "", string = "", string = "", int = 11111);//default constructor using default parameters

};

void welcomeScreen(); //Displays welcome screen, used at beginning of program

void displayFirstMenu();// Displays the opening menu for the program

void displayDeptSubMenu();//Displays the menu of options when a user selects Department menu in the first menu

void displayStaffSubMenu();//Displays the menu of options when a user selects Staff menu in the first menu

void displayPatientSubMenu();//Displays the menu of options when a user selects Patient menu in the first menu

void menuSelection(int);//Determines which sub-menu to display based on the user's choice in the first menu

string readCharArrayFromFiletoString(ifstream&);//Function used to read characters (including spaces) from a file with CSVs and return it as a string

void addADepartment(Department[], int&);//main function used to add a department to an array of type Department

void addAStaffMember(Staff[], Department[], int&, int);//main function used to add a staff member to an array of type Staff

void addAPatient(Patient[], Staff[], int&, int);//main function used to add a patient to an array of type Patient

void listAllDepartments(Department[], int);//main function used to list all departments in an array of type Department

void listAllStaff(Staff[], int);//main function used to list all staff members in an array of type Staff

void listAllPatients(Patient[], int);//main function used to list all patients in an array of type Patient

bool doesDeptIdExist(Department[], int, int );//similar to searchDept, but doesn't return an index (used only for checking if a department exists)

bool doesStaffMemberExist(Staff[], int, int);//checks if a staff member exists and is a doctor (used for adding patient function)

bool searchDept(Department[], int, int&, int);//returns true (and returns the array index where the ID is found) if a Department ID is found, false if it is not

bool searchStaffMember(Staff[], int, int&, int);//returns true (and returns the array index where the ID is found) if a staff member ID is found, false if it is not

bool searchPatient(Patient[], int, int&, int);//returns true (and returns the array index where the ID is found) if a patient ID is found, false if it is not

void subMenusFunction(int, int, Department[], Staff[],

Patient[], int&, int&, int&);//main function responsible for all submenu choices

void printAllPatientsToFile(Patient[], int, ofstream&);//prints all patients to an output file

void printAllStaffToFile(Staff[], int, ofstream&);//prints all staff members to an output file

void printAllDepartmentsToFile(Department[], int, ofstream&);//prints all departments to an output file

void departmentBubbleSort(Department[], int);//applies the bubblesort algorithm to an array of type Department

void staffBubbleSort(Staff[], int);//applies the bubblesort algorithm to an array of type Staff

void patientBubbleSort(Patient[], int);//applies the bubblesort algorithm to an array of type Patient

void editDepartment(Department[], int, int);//main function used in the edit option in the department submenu

void editStaffMember(Staff[], int, int, Department[], int);//main function used in the edit option in the staff submenu

void editPatient(Patient[], int, int, Staff[], int);//main function used in the edit option in the patient submenu

void outFileOperations(ofstream&, ofstream&, ofstream&, Department[], Staff[], Patient[], int, int, int);//performs all ofstream functions in the program

void inFileOperations(ifstream&, ifstream&, ifstream&, Department[], Staff[], Patient[], int&, int&, int&, int);//performs all ifstream functions in the program

Functions.cpp

//Author: Michael Hernandez

//BCS 230 Capstone Project: Midterm draft

//Date last modified: May 7, 2016

#include <string>

#include<iostream>

#include<fstream>

#include "Functions.h"

using namespace std;

void Department::readFromFile(ifstream& inFile) {

deptName = readCharArrayFromFiletoString(inFile);

inFile >> ID;

inFile.ignore();

}

void Department::printToFile(ofstream& outFile) {

outFile << deptName << "," << ID;

}

void Department::setDeptName(string departmentN) {

deptName = departmentN;

}

void Department::setDeptID(int departmentID) {

ID = departmentID;

}

int Department::getDeptID() const {

return ID;

}

string Department::getDeptName() const {

return deptName;

}

void Department::printDepartment() const {//tentative

cout << "Department Name: " << deptName << endl;

cout << "Department ID: " << ID << endl;

}

void listAllDepartments(Department departmentObject[], int numOfDepts) {

if (numOfDepts == 0) {

cout << "There are no departments listed in the registry." << endl << endl;

}

else {

departmentBubbleSort(departmentObject, numOfDepts);

for (int i = 0; i < numOfDepts; i++) {

departmentObject[i].printDepartment();

cout << endl;

}

}

}

Department::Department() {

deptName = "";

ID = 0;

}

void Staff::setStaffID(int sID) {

staffID = sID;

}

void Staff::setStaffType(char sType) {

staffType = sType;

}

int Staff::getStaffID() const {

return staffID;

}

void Staff::printStaff() const{

string jobType = "";

switch (staffType) {

case 'D':

jobType = "Doctor";

break;

case 'N':

jobType = "Nurse";

break;

case 'A':

jobType = "Administrator";

}

cout << "Name: " << jobType << " " << getFirstName() << " " << getLastName() << endl;

cout << "Staff ID: " << staffID << endl;

cout << "Department ID: " << staffDept.getDeptID() << endl;

cout << "Gender: " << getGender() << endl;

cout << "Birthday: " << getBirthMonth() << "/" << getBirthDay() << "/"

<< getBirthYear() << endl;

int numberOfPhones;

if (staffContact.getPhoneNoAmount() > 1)

numberOfPhones = staffContact.getPhoneNoAmount();

else

numberOfPhones = 1;

cout << "Phone Number: ";

for (int i = 0; i < numberOfPhones; i++) {

cout << staffContact.phoneNo[i] << endl;

}

int numberOfEmails;

if (staffContact.getEmailAmount() > 1)

numberOfEmails = staffContact.getEmailAmount();

else

numberOfEmails = 1;

cout << "E-mail: ";

for (int i = 0; i < numberOfEmails; i++) {

cout << staffContact.email[i] << endl;

}

}

void listAllStaff(Staff staffObject[], int numOfStaff) {

if (numOfStaff == 0) {

cout << "There are no staff members listed in the registry." << endl << endl;

}

else {

staffBubbleSort(staffObject, numOfStaff);

for (int i = 0; i < numOfStaff; i++) {

staffObject[i].printStaff();

cout << endl;

}

}

}

void Staff::readFromFile(ifstream& inFile) {

char c;//removes commas

char gender;

int birthD, birthM, birthY;

int departmentID;

if (inFile.good()) {

inFile >> staffType >> c;

setFirstName (readCharArrayFromFiletoString(inFile));

setLastName (readCharArrayFromFiletoString(inFile));

inFile >> staffID >> c >> departmentID >> c >> birthM >> c

>> birthD >> c >> birthY >> c

>> gender >> c;

staffContact.phoneNo[0] = readCharArrayFromFiletoString(inFile);

getline(inFile, staffContact.email[0]);

staffDept.setDeptID(departmentID);

setBirthDay(birthD);

setBirthMonth(birthM);

setBirthYear(birthY);

setGender(gender);

}

}

void Staff::printToFile(ofstream& outFile) {

outFile << staffType << ",";

outFile << getFirstName() << "," << getLastName() << ",";

outFile << staffID << ",";

outFile << staffDept.getDeptID() << ",";//department ID

outFile << getBirthMonth() << "/" << getBirthDay() << "/"

<< getBirthYear() << ",";

outFile << getGender() << ",";

outFile << staffContact.phoneNo[0] << ",";

outFile << staffContact.email[0];

}

char Staff::getStaffType() const {

return staffType;

}

Staff::Staff(int id, int deptID, char gen, int sDay, int sMonth, int sYear, int sEmailAm, int sPhoneAmount, string sEmail, string sPhoneNo, char sStaffType) {

staffID = id;

staffDept.setDeptID(deptID);

setGender(gen);

setBirthDay(sDay);

setBirthMonth(sMonth);

setBirthYear(sYear);

staffContact.setNumOfEmail(sEmailAm);

staffContact.setNumofPhones(sPhoneAmount);

staffContact.setAnEmail(sEmail, 0);

staffContact.setANumber(sPhoneNo, 0);

staffType = sStaffType;

}

void Patient::setID(int patientIdent) {

patientID = patientIdent;

}

void Patient::setBirthday(int pbirthDay, int pbirthMonth, int pbirthYear) {

setBirthDay (pbirthDay);

setBirthMonth (pbirthMonth);

setBirthYear (pbirthYear);

}

int Patient::getPatientID() const {

return patientID;

}

void Patient::printPatientInfo() const{//List All

cout << "Name: " << getFirstName() << " " << getLastName() << endl;

cout << "Patient ID: " << patientID << endl;

cout << "Gender: " << getGender() << endl;

cout << "Birthday: " << getBirthMonth() << "/" << getBirthDay() << "/"

<< getBirthYear() << endl;

cout << "Phone Number: "<< patientContact.phoneNo << endl;

cout << "Email: " << patientContact.email << endl;

cout << "Address: " << patientContact.patientAddress.addNo << " " << patientContact.patientAddress.streetName << endl;

cout << patientContact.patientAddress.cityName << ", " << patientContact.patientAddress.stateName << " "

<< patientContact.patientAddress.zipCode << endl;

}

void listAllPatients(Patient patientObject[], int numOfPatients) {

if (numOfPatients == 0) {

cout << "There are no patients listed in the registry." << endl << endl;

}

else {

patientBubbleSort(patientObject, numOfPatients);

for (int i = 0; i < numOfPatients; i++) {

patientObject[i].printPatientInfo();

cout << endl;

}

}

}

Patient::Patient(int pID, int sID, char pGender, int pDay, int pMonth, int pYear, string pFirst, string pLast, string pEmail, string pPhoneNo, int pAddressNo, string pStreetName, string pCityName, string pStateName, int pZipCode)

:Person(pGender, pMonth, pDay, pYear, pFirst, pLast)

{

patientID = pID;

attendingStaffID = sID;

patientContact.email = pEmail;

patientContact.phoneNo = pPhoneNo;

patientContact.patientAddress.addNo = pAddressNo;

patientContact.patientAddress.cityName = pCityName;

patientContact.patientAddress.streetName = pStreetName;

patientContact.patientAddress.stateName = pStateName;

patientContact.patientAddress.zipCode = pZipCode;

}

void Patient::readFromFile(ifstream& inFile) {

char c;//eliminates commas

char gender;

int birthDay, birthMonth, birthYear;

if (inFile.good()) {

setFirstName(readCharArrayFromFiletoString(inFile));

setLastName(readCharArrayFromFiletoString(inFile));

inFile >> patientID >> c >> attendingStaffID >> c >> birthMonth >> c >> birthDay;

inFile >> c >> birthYear >> c >> gender >> c >> patientContact.patientAddress.addNo >> c;

patientContact.patientAddress.streetName = readCharArrayFromFiletoString(inFile);

patientContact.patientAddress.cityName = readCharArrayFromFiletoString(inFile);

patientContact.patientAddress.stateName = readCharArrayFromFiletoString(inFile);

inFile >> patientContact.patientAddress.zipCode >> c;

patientContact.phoneNo = readCharArrayFromFiletoString(inFile);

getline(inFile, patientContact.email);

setBirthMonth(birthMonth);

setBirthDay(birthDay);

setBirthYear(birthYear);

setGender(gender);

}

}

void Patient::printToFile(ofstream& outFile) {

outFile << getFirstName() << "," << getLastName() << ",";

outFile << patientID << ",";

outFile << getAttendingID() << ",";

outFile << getBirthMonth() << "/" << getBirthDay() << "/"

<< getBirthYear()<< ",";

outFile << getGender() << ",";

outFile << patientContact.patientAddress.addNo << "," << patientContact.patientAddress.streetName << ",";

outFile << patientContact.patientAddress.cityName << "," << patientContact.patientAddress.stateName << ","

<< patientContact.patientAddress.zipCode << ",";

outFile << patientContact.phoneNo << ",";

outFile << patientContact.email;

}

void Patient::setAddress(int houseNum, string street, string city, string stateInit, int zipNum) {

patientContact.patientAddress.addNo = houseNum;

patientContact.patientAddress.streetName = street;

patientContact.patientAddress.cityName = city;

patientContact.patientAddress.stateName = stateInit;

patientContact.patientAddress.zipCode = zipNum;

}

void Patient::setPhoneNum(string phone) {

patientContact.phoneNo = phone;

}

void Patient::setEmail(string pEmail) {

patientContact.email = pEmail;

}

void Patient::setPatientContactInfo(PatientContactInfo newPatientContact) {

patientContact = newPatientContact;

}

void Person::setFirstName(string personFirstName) {

firstName = personFirstName;

}

void Person::setLastName(string personLastName) {

lastName = personLastName;

}

void Person::setGender(char personGender) {

gender = personGender;

}

void Person::setBirthDay(int bDay){

birthD.day = bDay;

}

void Person::setBirthMonth(int bMonth) {

birthD.month = bMonth;

}

void Person::setBirthYear(int bYear) {

birthD.year = bYear;

}

int Person::getBirthDay()const {

return birthD.day;

}

int Person::getBirthMonth()const {

return birthD.month;

}

int Person::getBirthYear() const {

return birthD.year;

}

char Person::getGender() const {

return gender;

}

string Person::getFirstName() const {

return firstName;

}

string Person::getLastName() const {

return lastName;

}

void displayFirstMenu() {

cout << "1. Department menu" << endl;

cout << "2. Staff menu" << endl;

cout << "3. Patient menu" << endl;

cout << "4. Exit" << endl;

}

void welcomeScreen() {

cout << "Welcome to the hospital menu program!" << endl << endl;

cout << "The purpose of this program is to store and present information about your hospital." << endl;

cout << "Below is the main menu, Enter 1-3 to select one of the options to be taken to a different section of the program." << endl

<<"To exit the program from the main menu Enter the number 4" << endl << endl;

}

void menuSelection(int theChoice){

switch (theChoice) {

case 1:

displayDeptSubMenu();

break;

case 2:

displayStaffSubMenu();

break;

case 3:

displayPatientSubMenu();

break;

default:

cout << "Invalid Selection. Try again." << endl;

break;

}

}

void displayDeptSubMenu() {

cout << "1. List all departments" << endl;

cout << "2. Add a department" << endl;

cout << "3. Search for a department" << endl;

cout << "4. Edit a department's information" << endl;

cout << "6. Return to main menu" << endl;

}

void displayStaffSubMenu() {

cout << "1. List all employees" << endl;

cout << "2. Add an employee" << endl;

cout << "3. Search for an employee" << endl;

cout << "4. Edit an employee's information" << endl;

cout << "6. Return to main menu" << endl;

}

void displayPatientSubMenu() {

cout << "1. List all patients" << endl;

cout << "2. Add a patient" << endl;

cout << "3. Search for a patient" << endl;

cout << "4. Edit a patient's information" << endl;

cout << "6. Return to main menu" << endl;

}

void subMenusFunction(int choice, int subchoice, Department departmentObject[], Staff staffObject[], Patient patientObject[], int &numOfDepts, int &numOfStaff, int &numOfPatients){

int newSearchID;

int idForPrint = 0;

switch (choice) {

case 1:

switch (subchoice) {

case 1:

listAllDepartments(departmentObject, numOfDepts);

break;

case 2:

addADepartment(departmentObject, numOfDepts);

cout << endl;

break;

case 3: {

cout << "Enter department ID: ";

cin >> newSearchID;

cout << endl;

bool isFound = searchDept(departmentObject, numOfDepts, idForPrint, newSearchID);

if (isFound) {

departmentObject[idForPrint].printDepartment();

cout << endl;

}

else {

cout << "Department not found." << endl << endl;

}

break;

}

case 4:

cout << "Enter the department ID or the department you wish to edit: ";

cin >> newSearchID;

cout << endl;

editDepartment(departmentObject, numOfDepts, newSearchID);

break;

default:

cout << "Invalid Selection. Please try again." << endl << endl;

}

break;

case 2:

switch (subchoice) {

case 1:

listAllStaff(staffObject, numOfStaff);

break;

case 2:

addAStaffMember(staffObject, departmentObject, numOfStaff, numOfDepts);

cout << endl;

break;

case 3: {

cout << "Enter staff member ID: ";

cin >> newSearchID;

cout << endl;

bool isFound = searchStaffMember(staffObject, numOfStaff, idForPrint, newSearchID);

if (isFound) {

staffObject[idForPrint].printStaff();

cout << endl;

int newDeptIndex = staffObject[idForPrint].staffDept.getDeptID();

bool isDeptFound = searchDept(departmentObject, numOfDepts, idForPrint, newDeptIndex);

if (isDeptFound) {

cout << "Department Information:" << endl << endl;

departmentObject[idForPrint].printDepartment();

cout << endl;

}

}

else {

cout << "Staff member not found." << endl << endl;

}

break;

}

case 4:

cout << "Enter the Staff ID of the staff member whose information you wish to edit: ";

cin >> newSearchID;

cout << endl;

editStaffMember(staffObject, numOfStaff, newSearchID, departmentObject, numOfDepts);

break;

default:

cout << "Invalid selection. Please try again." << endl << endl;

break;

}

break;

case 3:

switch (subchoice) {

case 1:

listAllPatients(patientObject, numOfPatients);

break;

case 2:

addAPatient(patientObject, staffObject, numOfPatients, numOfStaff);

cout << endl;

break;

case 3: {

cout << "Enter Patient ID: ";

cin >> newSearchID;

cout << endl;

bool isFound = searchPatient(patientObject, numOfPatients, idForPrint, newSearchID);

if (isFound) {

patientObject[idForPrint].printPatientInfo();

cout << endl;

int newDoctorID = patientObject[idForPrint].getAttendingID();

bool isDoctorFound = searchStaffMember(staffObject, numOfStaff, idForPrint, newDoctorID);

if (isDoctorFound) {

cout << "Attending Staff Member Information: " << endl << endl;

staffObject[idForPrint].printStaff();

cout << endl;

}

}

else {

cout << "Patient not found." << endl << endl;

}

break;

}

case 4:

cout << "Enter the ID of the patient whose information you wish to edit: ";

cin >> newSearchID;

cout << endl;

editPatient(patientObject, numOfPatients, newSearchID, staffObject, numOfStaff);

break;

default:

cout << "Invalid selection. Please try again." << endl << endl;

}

break;

}

}

string readCharArrayFromFiletoString(ifstream& inFile) {

char c;

int characterCount = 0;

char name[100];

while (inFile.get(c)) {

if (c == ',')

break;

name[characterCount] = c;

characterCount++;

}

string nameChar(name, characterCount);

return nameChar;

}

void addADepartment(Department departmentObject[], int &departmentSize) {

string deptName;

int deptID;

cout << "Enter the following information about the department you want to add." << endl;

cout << "Name: ";

cin.ignore();

getline(cin, deptName);

departmentObject[departmentSize].setDeptName(deptName);

cout << endl;

cout << "ID: ";

cin >> deptID;

departmentObject[departmentSize].setDeptID(deptID);

departmentSize++;

}

void addAStaffMember(Staff staffObject[], Department departmentObject[], int &staffSize, int numOfDepartments) {

int newStaffID;

int newDeptID;

char newStaffType;

cout << "Enter the following information about the staff member you wish to add." << endl;

cout << "Staff ID: ";

cin >> newStaffID;

cout << endl;

staffObject[staffSize].setStaffID(newStaffID);

cout << "Staff Type(enter A for administrator, D for doctor, or N for nurse): ";

cin >> newStaffType;

cout << endl;

cout << "Department ID: ";

cin >> newDeptID;

cout << endl;

staffObject[staffSize].staffDept.setDeptID(newDeptID);

while (!doesDeptIdExist(departmentObject, staffObject[staffSize].staffDept.getDeptID(), numOfDepartments)) {

cout << "Invalid Department ID. Please reenter an existing Department ID: ";

cin >> newDeptID;

cout << endl;

staffObject[staffSize].staffDept.setDeptID(newDeptID);

}

string fName;

cout << "First Name: ";

cin >> fName;

cout << endl;

string lName;

cout << "Last Name: ";

cin >> lName;

cout << endl;

char gender;

cout << "Gender (M or F): ";

cin >> gender;

cout << endl;

int bDay, bMonth, bYear;

cout << "Birthday (month day year) seperated by spaces: ";

cin >> bMonth >> bDay >> bYear;

cout << endl;

cout << "How many phone numbers would you like to store?: ";

int phoneNumAmount;

cin >> phoneNumAmount;

staffObject[staffSize].staffContact.setNumofPhones(phoneNumAmount);

string newNumber = "";

cin.ignore();

for (int i = 0; i < staffObject[staffSize].staffContact.getPhoneNoAmount(); i++) {

if(i == 0)

cout << "Enter Phone number: ";

else {

cout << "Enter Phone number #" << i + 1 << ": ";

}

getline(cin, newNumber);

cout << endl;

staffObject[staffSize].staffContact.setANumber(newNumber, i);

}

cout << "How many E-mail addresses would you like to store?: ";

int howManyEmail;

cin >> howManyEmail;

staffObject[staffSize].staffContact.setNumOfEmail(howManyEmail);

string newEmail = "";

cin.ignore();

for (int i = 0; i < staffObject[staffSize].staffContact.getEmailAmount(); i++) {

if(i == 0)

cout << "E-mail: ";

else {

cout << "E-mail #" << i + 1 << ": ";

}

getline(cin, newEmail);

cout << endl;

staffObject[staffSize].staffContact.setAnEmail(newEmail, i);

}

staffObject[staffSize].setStaffType(newStaffType);

staffObject[staffSize].setFirstName(fName);

staffObject[staffSize].setLastName(lName);

staffObject[staffSize].setGender(gender);

staffObject[staffSize].setBirthDay(bDay);

staffObject[staffSize].setBirthMonth(bMonth);

staffObject[staffSize].setBirthYear(bYear);

staffSize++;

}

void addAPatient(Patient patientObject[], Staff staffObject[], int &patientIndex, int numOfStaff) {

Person newPatient;

PatientContactInfo newPatientContact;

int newPatientID;

int patientAttendingStaffID;

cout << "Enter the following information about the patient you wish to add." << endl;

string fName;

cout << "First Name: ";

cin >> fName;

cout << endl;

string lName;

cout << "Last Name: ";

cin >> lName;

cout << endl;

cout << "Patient ID: ";

cin >> newPatientID;

cout << endl;

cout << "Doctor ID: ";

cin >> patientAttendingStaffID;

cout << endl;

patientObject[patientIndex].setAttendingID(patientAttendingStaffID);

while (!doesStaffMemberExist(staffObject, patientObject[patientIndex].getAttendingID(), numOfStaff)) {

cout << "There is no doctor with that ID. Please enter the ID of an existing doctor: ";

cin >> patientAttendingStaffID;

cout << endl;

patientObject[patientIndex].setAttendingID(patientAttendingStaffID);

}

char gender;

cout << "Gender(M or F): ";

cin >> gender;

cout << endl;

cout << "Phone Number: ";

cin >> newPatientContact.phoneNo;

cout << endl;

cout << "E-mail: ";

cin >> newPatientContact.email;

cout << endl;

int bDay, bMonth, bYear;

cout << "Birth Date(month day year) seperated by spaces: ";

cin >> bMonth >> bDay >> bYear;

cout << endl;

cout << "Address Number: ";

cin >> newPatientContact.patientAddress.addNo;

cout << endl;

cout << "Street Name: ";

cin.ignore();

getline(cin, newPatientContact.patientAddress.streetName);

cout << endl;

cout << "City Name: ";

getline(cin, newPatientContact.patientAddress.cityName);

cout << endl;

cout << "State Initials: ";

getline(cin, newPatientContact.patientAddress.stateName);

cout << endl;

cout << "Zip Code: ";

cin >> newPatientContact.patientAddress.zipCode;

cout << endl;

patientObject[patientIndex].setGender(gender);

patientObject[patientIndex].setFirstName(fName);

patientObject[patientIndex].setLastName(lName);

patientObject[patientIndex].setPatientContactInfo(newPatientContact);

patientObject[patientIndex].setID(newPatientID);

patientObject[patientIndex].setBirthDay(bDay);

patientObject[patientIndex].setBirthMonth(bMonth);

patientObject[patientIndex].setBirthYear(bYear);

patientIndex++;

}

bool doesDeptIdExist(Department departmentObject[], int thisDeptId, int numOfDepartments){

bool deptExist = false;

for (int i = 0; i < numOfDepartments; i++) {

int newDeptId = departmentObject[i].getDeptID();

if (thisDeptId == newDeptId) {

deptExist = true;

break;

}

}

return deptExist;

}

bool doesStaffMemberExist(Staff staffObject[], int thisStaffId, int numOfStaff) {

bool staffExist = false;

for (int i = 0; i < numOfStaff; i++) {

int newStaffId = staffObject[i].getStaffID();

if (thisStaffId == newStaffId && staffObject[i].getStaffType() == 'D') {

staffExist = true;

break;

}

}

return staffExist;

}

bool searchDept(Department departmentObject[], int numOfDepts, int &foundIndex, int searchID) {

bool found = false;

for (int i = 0; i < numOfDepts; i++) {

int newDeptID = departmentObject[i].getDeptID();

if (searchID == newDeptID) {

found = true;

foundIndex = i;

break;

}

}

return found;

}

bool searchStaffMember(Staff staffObject[], int numOfStaff, int &foundIndex, int searchID) {

bool found = false;

for (int i = 0; i < numOfStaff; i++) {

int newStaffID = staffObject[i].getStaffID();

if (searchID == newStaffID) {

found = true;

foundIndex = i;

break;

}

}

return found;

}

bool searchPatient(Patient patientObject[], int numOfPatients, int &foundIndex, int searchID) {

bool found = false;

for (int i = 0; i < numOfPatients; i++) {

int newPatientID = patientObject[i].getPatientID();

if (searchID == newPatientID) {

found = true;

foundIndex = i;

break;

}

}

return found;

}

void printAllDepartmentsToFile(Department departmentObject[], int numOfDepartments, ofstream& outFile) {

for (int i = 0; i < numOfDepartments; i++) {

departmentObject[i].printToFile(outFile);

outFile << endl;

}

}

void printAllStaffToFile(Staff staffObject[], int numOfStaff, ofstream& outFile) {

for (int i = 0; i < numOfStaff; i++) {

staffObject[i].printToFile(outFile);

outFile << endl;

}

}

void printAllPatientsToFile(Patient patientObject[], int numOfPatients, ofstream& outFile) {

for (int i = 0; i < numOfPatients; i++) {

patientObject[i].printToFile(outFile);

outFile << endl;

}

}

void departmentBubbleSort(Department departmentObject[], int numOfDepartments) {

bool isSwapped = true;

int l = 0;

Department tempDept;

while (isSwapped) {

isSwapped = false;

l++;

for (int i = 0; i < numOfDepartments - l; i++) {

if (departmentObject[i].getDeptID() > departmentObject[i + 1].getDeptID()) {

tempDept = departmentObject[i];

departmentObject[i] = departmentObject[i + 1];

departmentObject[i + 1] = tempDept;

isSwapped = true;

}

}

}

}

void staffBubbleSort(Staff staffObject[], int numOfStaff) {

bool isSwapped = true;

int l = 0;

Staff tempStaff;

while (isSwapped) {

isSwapped = false;

l++;

for (int i = 0; i < numOfStaff - l; i++) {

if (staffObject[i].getStaffID() > staffObject[i + 1].getStaffID()) {

tempStaff = staffObject[i];

staffObject[i] = staffObject[i + 1];

staffObject[i + 1] = tempStaff;

isSwapped = true;

}

}

}

}

void patientBubbleSort(Patient patientObject[], int numOfPatients) {

bool isSwapped = true;

int l = 0;

Patient tempStorage;

while (isSwapped) {

isSwapped = false;

l++;

for (int i = 0; i < numOfPatients - l; i++) {

if (patientObject[i].getPatientID() > patientObject[i + 1].getPatientID()) {

tempStorage = patientObject[i];

patientObject[i] = patientObject[i + 1];

patientObject[i + 1] = tempStorage;

isSwapped = true;

}

}

}

}

void editDepartment(Department departmentObject[], int numOfDepts, int searchID) {

int departmentIndex = 0;

bool foundDepartment = searchDept(departmentObject, numOfDepts, departmentIndex, searchID);

if (foundDepartment) {

int deptChoice;

string newName = "";

int newID = 0;

cout << "Select the option that corresponds to what you want to edit about this department:" << endl;

cout << "1) Department Name" << endl << "2) Department ID" << endl;

cin >> deptChoice;

cout << endl;

switch (deptChoice) {

case 1:

cout << "Enter the new name for this department: ";

cin >> newName;

cout << endl;

departmentObject[departmentIndex].setDeptName(newName);

break;

case 2:

cout << "Enter the new ID number for this department: ";

cin >> newID;

cout << endl;

departmentObject[departmentIndex].setDeptID(newID);

break;

default:

cout << "Invalid Selection." << endl;

}

}

else {

cout << "Department does not exist." << endl;

}

}

void editStaffMember(Staff staffObject[], int numOfStaff, int searchID, Department deptObject[], int numOfDepartments) {

int staffIndex = 0;

bool foundStaffMember = searchStaffMember(staffObject, numOfStaff, staffIndex, searchID);

if (foundStaffMember) {

int staffChoice;

string newInfo = "";

int newID = 0;

int day = 1, month = 1, year = 1;

char newIdentifier = ' ';

cout << "Select the option that corresponds to the information you want to edit" << endl;

cout << "1) First Name" << endl << "2) Last Name" << endl << "3) Staff ID" << endl << "4) Staff Type"

<< endl << "5) Gender" << endl << "6) Birthday" << endl << "7) Phone Number" << endl

<< "8) E-mail" << endl << "9) Department ID" << endl;

cin >> staffChoice;

cout << endl;

switch(staffChoice){

case 1:

cout << "Enter the first name of this staff member: ";

cin >> newInfo;

cout << endl;

staffObject[staffIndex].setFirstName(newInfo);

break;

case 2:

cout << "Enter the last name of this staff member: ";

cin >> newInfo;

cout << endl;

staffObject[staffIndex].setLastName(newInfo);

break;

case 3:

cout << "Enter new Staff ID: ";

cin >> newID;

cout << endl;

staffObject[staffIndex].setStaffID(newID);

break;

case 4:

cout << "Enter A if the staff member is an administrator, N if the staff member is a nurse or D if the staff member is a doctor: ";

cin >> newIdentifier;

cout << endl;

staffObject[staffIndex].setStaffType(newIdentifier);

break;

case 5:

cout << "Enter M if the staff member is a male or enter F if the staff member is a female: ";

cin >> newIdentifier;

cout << endl;

staffObject[staffIndex].setGender(newIdentifier);

break;

case 6:

cout << "Enter the birthday (month day year) of the staff member, be sure to seperate each number with a space: ";

cin >> month >> day >> year;

cout << endl;

staffObject[staffIndex].setBirthMonth(month);

staffObject[staffIndex].setBirthDay(day);

staffObject[staffIndex].setBirthYear(year);

break;

case 7: {

cout << "How many phone numbers would you like to store for this staff member?: ";

int staffPhoneNumAm;

cin >> staffPhoneNumAm;

staffObject[staffIndex].staffContact.setNumofPhones(staffPhoneNumAm);

cin.ignore();

for (int i = 0; i < staffPhoneNumAm; i++) {

if (i == 0)

cout << "Enter new phone number: ";

else

cout << "Enter phone number #" << i + 1 << ": ";

getline(cin, newInfo);

cout << endl;

staffObject[staffIndex].staffContact.setANumber(newInfo, i);

}

break;

}

case 8: {

cout << "How many E-mails would you like to store for this staff member?: ";

int newEmailNum;

cin >> newEmailNum;

cout << endl;

staffObject[staffIndex].staffContact.setNumOfEmail(newEmailNum);

cin.ignore();

for (int i = 0; i < newEmailNum; i++) {

if (i == 0)

cout << "Enter E-mail: ";

else

cout << "Enter E-mail #" << i + 1 << ": ";

getline(cin, newInfo);

cout << endl;

staffObject[staffIndex].staffContact.setAnEmail(newInfo, i);

}

break;

}

case 9: {

int tempDeptID;

cout << "Enter Department ID: ";

cin >> tempDeptID;

while (!doesDeptIdExist(deptObject, tempDeptID, numOfDepartments)) {

cout << "That department does not exist.\nPlease enter an existing department: ";

cin >> tempDeptID;

cout << endl;

}

staffObject[staffIndex].staffDept.setDeptID(tempDeptID);

break;

}

default:

cout << "Invalid Selection." << endl;

}

}

else {

cout << "Staff member not found." << endl;

}

}

void editPatient(Patient patientObject[], int numOfPatients, int searchID, Staff staffObject[], int numOfStaff) {

int patientIndex = 0;

bool foundPatient = searchPatient(patientObject, numOfPatients, patientIndex, searchID);

if (foundPatient){

int patientChoice;

string newInfo = "";

string newStreet = "", newCity = "", newState = "";

int newHouse = 0, newZip = 0;

PatientContactInfo newPatientContact;

char newIdentifier = ' ';

int day = 1, month = 1, year = 1;

int newID = 0;

cout << "Select the option that corresponds to the information you want to edit" << endl;

cout << "1) First Name" << endl << "2) Last Name" << endl << "3) Patient ID" << endl << "4) Home Address"

<< endl << "5) Gender" << endl << "6) Birthday" << endl << "7) Phone Number" << endl

<< "8) E-mail" << endl << "9) Doctor ID" << endl;

cin >> patientChoice;

cout << endl;

switch (patientChoice) {

case 1:

cout << "Enter the patient's first name: ";

cin >> newInfo;

cout << endl;

patientObject[patientIndex].setFirstName(newInfo);

break;

case 2:

cout << "Enter the patient's last name: ";

cin >> newInfo;

cout << endl;

patientObject[patientIndex].setLastName(newInfo);

break;

case 3:

cout << "Enter the patient's new ID number: ";

cin >> newID;

cout << endl;

patientObject[patientIndex].setID(newID);

break;

case 4:

cout << "Enter the following information regarding the patient's home address: " << endl;

cout << "House Number: ";

cin >> newHouse;

cout << endl;

cout << "Street Name: ";

cin.ignore();

getline(cin, newStreet);

cout << endl;

cout << "City Name: ";

getline(cin, newCity);

cout << endl;

cout << "State Initials: ";

getline(cin, newState);

cout << endl;

cout << "Zip code: ";

cin >> newZip;

cout << endl;

patientObject[patientIndex].setAddress(newHouse, newStreet, newCity, newState, newZip);

break;

case 5:

cout << "Enter the patient's gender(M for male or F for female): ";

cin >> newIdentifier;

cout << endl;

patientObject[patientIndex].setGender(newIdentifier);

break;

case 6:

cout << "Enter the birthday (month day year) of the patient, be sure to seperate each number with a space: ";

cin >> month >> day >> year;

cout << endl;

patientObject[patientIndex].setBirthDay(day);

patientObject[patientIndex].setBirthMonth(month);

patientObject[patientIndex].setBirthYear(year);

break;

case 7:

cout << "Enter the patient's phone number: ";

cin >> newInfo;

cout << endl;

patientObject[patientIndex].setPhoneNum(newInfo);

break;

case 8:

cout << "Enter the patient's E-mail: ";

cin >> newInfo;

cout << endl;

patientObject[patientIndex].setEmail(newInfo);

break;

case 9: {

int tempStaffID;

cout << "Enter the ID of the Attending Doctor: ";

cin >> tempStaffID;

cout << endl;

while(!doesStaffMemberExist(staffObject, tempStaffID, numOfStaff)){

cout << "There is no Doctor with that ID.\nPlease enter the ID of the Attending Doctor: ";

cin >> tempStaffID;

cout << endl;

}

patientObject[patientIndex].setAttendingID(tempStaffID);

break;

}

default:

cout << "Invalid Selection." << endl;

}

}

else {

cout << "Patient not found." << endl;

}

}

StaffContactInfo::StaffContactInfo(int thePhone, int theEmail) {

emailAmount = theEmail;

phoneNoAmount = thePhone;

email = new string[theEmail];

phoneNo = new string[thePhone];

}

StaffContactInfo::~StaffContactInfo() {

delete[] phoneNo;

delete[] email;

}

void StaffContactInfo::setNumOfEmail(int numOfEmail) {

emailAmount = numOfEmail;

email = new string[numOfEmail];

}

void StaffContactInfo::setNumofPhones(int phoneNumbers) {

phoneNoAmount = phoneNumbers;

phoneNo = new string[phoneNumbers];

}

int StaffContactInfo::getPhoneNoAmount() const {

return phoneNoAmount;

}

void StaffContactInfo::setANumber(string newPNum, int phoneIndex) {

phoneNo[phoneIndex] = newPNum;

}

int StaffContactInfo::getEmailAmount() const{

return emailAmount;

}

void StaffContactInfo::setAnEmail(string newEmail, int emailIndex) {

email[emailIndex] = newEmail;

}

StaffContactInfo::StaffContactInfo(const StaffContactInfo& otherObject) {

emailAmount = otherObject.emailAmount;

phoneNoAmount = otherObject.phoneNoAmount;

phoneNo = new string[phoneNoAmount];

for (int i = 0; i < phoneNoAmount; i++) {

phoneNo[i] = otherObject.phoneNo[i];

}

email = new string[emailAmount];

for (int i = 0; i < emailAmount; i++) {

email[i] = otherObject.email[i];

}

}

const StaffContactInfo& StaffContactInfo::operator= (const StaffContactInfo& myObject) {

if (this != &myObject) {

delete[] email;

delete[] phoneNo;

emailAmount = myObject.emailAmount;

phoneNoAmount = myObject.phoneNoAmount;

email = new string[emailAmount];

phoneNo = new string[phoneNoAmount];

for (int i = 0; i < emailAmount; i++)

email[i] = myObject.email[i];

for (int l = 0; l < phoneNoAmount; l++)

phoneNo[l] = myObject.phoneNo[l];

}

return \*this;

}

void Patient::setAttendingID(int pAttend){

attendingStaffID = pAttend;

}

int Patient::getAttendingID() const {

return attendingStaffID;

}

Person::Person(char pGender, int pBmonth, int pBday, int pByear, string pfName, string plName) {

gender = pGender;

birthD.day = pBday;

birthD.month = pBmonth;

birthD.year = pByear;

firstName = pfName;

lastName = plName;

}

void outFileOperations(ofstream& outFile1, ofstream& outFile2, ofstream& outFile3, Department dList[], Staff sList[], Patient pList[], int deptNum, int staffNum, int patNum) {

outFile1.open("updated\_departments.txt");

outFile2.open("updated\_staff.txt");

outFile3.open("updated\_patients.txt");

printAllDepartmentsToFile(dList, deptNum, outFile1);

printAllStaffToFile(sList, staffNum, outFile2);

printAllPatientsToFile(pList, patNum, outFile3);

outFile1.close();

outFile2.close();

outFile3.close();

}

void inFileOperations(ifstream& dInFile, ifstream& sInFile, ifstream& pInFile, Department dList[], Staff sList[], Patient pList[], int& deptNum, int& staffNum, int& patNum, int maximum) {

dInFile.open("departments.txt");

sInFile.open("staff.txt");

pInFile.open("patients.txt");

while (dInFile.good()) {

if (dInFile.peek() == ifstream::traits\_type::eof())

break;

if (deptNum >= maximum)

break;

dList[deptNum].readFromFile(dInFile);

deptNum++;

}

while (sInFile.good()) {

if (sInFile.peek() == ifstream::traits\_type::eof())

break;

if (staffNum >= maximum)

break;

sList[staffNum].readFromFile(sInFile);

staffNum++;

}

while (pInFile.good()) {

if (pInFile.peek() == ifstream::traits\_type::eof())

break;

if (patNum >= maximum)

break;

pList[patNum].readFromFile(pInFile);

patNum++;

}

dInFile.close();

sInFile.close();

pInFile.close();

}

Main.cpp

//Author: Michael Hernandez

//BCS 230 Capstone Project: Midterm draft

//Date last modified: May 7, 2016

#include<iostream>

#include<fstream>

#include<string>

#include "Functions.h"

using namespace std;

int main() {

const int MAX\_SIZE = 100;//stores maximum size for the department, staff, and patient arrays

int choice, subchoice;//used to store user choices in the main menu and submenus

ifstream inFiled, inFilep, inFiles;//used to perform operations on input files

ofstream outFileD, outFileP, outFileS;//used to perform operations on output files

Department departmentList[MAX\_SIZE];//stores information on up to the MAX\_SIZE of departments

Staff staffList[MAX\_SIZE];//stores information on up to the MAX\_SIZE of staff members

Patient patientList[MAX\_SIZE];//stores information on up to the MAX\_SIZE of patients

int numOfDepartments = 0;//keeps track of the number of departments stored in departmentList for use in the program

int numOfStaff = 0;//keeps track of the number of staff members stored in staffList for use in the program

int numOfPatients = 0;//keeps track of the number of staff members stored in patientList for use in the program

inFileOperations(inFiled, inFiles, inFilep, departmentList, staffList, patientList, numOfDepartments, numOfStaff, numOfPatients, MAX\_SIZE);// infile operations

welcomeScreen();//displays welcome screen for program

displayFirstMenu();//displays the main menu for the first time

cin >> choice;//takes in user choice for the first time

while (choice != 4) {

cout << endl;//formatting

menuSelection(choice);

if (choice < 4 && choice > 0)

cin >> subchoice;

cout << endl;//formatting

while(subchoice != 6){

subMenusFunction(choice, subchoice, departmentList, staffList, patientList, numOfDepartments, numOfStaff, numOfPatients);

menuSelection(choice);

cin >> subchoice;

}

displayFirstMenu();

cin >> choice;

} //main menu loop with nested submenus

outFileOperations(outFileD, outFileS, outFileP, departmentList, staffList, patientList, numOfDepartments, numOfStaff, numOfPatients);//outfile operations

cout << "Thank you for using the program! Goodbye!" << endl;//closing message

system("pause");

return 0;

}