// C++ Doctor Program.cpp : Defines the entry point for the console application.

//

/\*

Program Name: C++ Doctor Program

Module Code: UFCFME-30-2 Object Oriented Software Design and Development

Developer: WS243476

Created: 28/02/2018

Last Modified: 18/03/2018

\*/

#include "stdafx.h"

#include "Accounts.h"

#include "ErrorCheck.h"

#include "System.h"

#include "Booking.h"

#include "DeleteObjects.h"

#include <iostream>

#include <string>

#include <array>

#include <vector>

#include <conio.h>

#include <ctime>

using namespace std;

int main()

{

// VARIABLES

\_\_int16 iNumberOfReceptionist = 0; // \_\_int16 to store numbers from –32,768 to 32,767, Number of receptionist accounts

\_\_int16 iNumberOfDoctors = 0; // Number of doctor accounts

\_\_int16 iNumberOfSurgerys = 0; // Number of surgery objects

\_\_int16 iFail = 0; // Used for login and lockout functionality

\_\_int16 iChoice = 0; // Used for menu options

\_\_int16 iPreviousChoice = 0; // Used for providing correct menu options

\_\_int16 iMenuCount = 0; // Used for menu loop counting to prevent booking crashes

\_\_int16 iDoctorChoice = 0; // Used for booking a doctor with a customer

\_\_int16 iSurgeryChoice = 0; // Used for booking a surgery with a customer

\_\_int16 iPreviousCustomerID = 0; // Previous customer ID

\_\_int16 iPreviousBookingID = 0; // Previous booking ID

\_\_int16 iPreviousDoctorID = 0; // Previous doctor ID

\_\_int16 iDoctorID = 0; // Doctor ID used for booking

\_\_int16 iBookingDoctorID = 0; // The last booking made within the system doctor ID

\_\_int16 iBookingSurgeryID = 0; // The last surgeyr made within the system surgery ID

\_\_int16 iSurgeryObjectSize; // Stores how many objects are within the surgery vector

\_\_int16 iDoctorObjectSize; // Stores how many objects are within the doctor vector

\_\_int16 iReceptionistObjectSize; // Stores how many objects are within the receptionist vector

\_\_int16 iCustomerObjectSize; // Stores how many objects are within the customer vector

\_\_int16 iBookingObjectSize = 0; // Stores how many objects are within the booking vector

\_\_int16 iObjectSize = 0; // Stores object size for checking if there is at least 1 object left before deleting

string sUsername = ""; // Username

string sPassword = ""; // Password

string sDoctorName; // Doctor name used for booking

string sCustomerName; // Customer name used for booking

string sCustomerAilment; // Customer ailment used for booking

string sBookingDate; // Used for booking dates

bool bCheck = false; // Error checking

vector<Receptionist> oReceptionist(1); // Receptionist vector

vector<Doctor> oDoctor(1); // Doctor vector

vector<Surgery>oSurgery(1); // Surgery vector

vector<Customer> oCustomer; // Customer vector

vector<Booking>oBooking; // Booking vector

System oSystem; // System object

Customer oNewCustomer; // Create new customer object

Booking oNewBooking; // Create new booking object

Doctor oNewDoctor; // Create new doctor object

Surgery oNewSurgery; // Create new surgery object

Receptionist oNewReceptionist; // Create receptionist object

time\_t tNow; // For the current time in the system

// MAIN PROGRAM

// Displays welcome message

oSystem.Welcome();

// Receptionist account creation

// Calls and assigns number of receptionists

iNumberOfReceptionist = oReceptionist[0].GetNumberOfObjects("receptionist");

// Create vector of receptionists to the user's inputted amount

oReceptionist.resize(iNumberOfReceptionist);

// Get number of receptionist accounts and enter username and password for each of them

for (\_\_int16 iCount = 0; iCount < iNumberOfReceptionist; ++iCount)

{

// Formatting for if more than 1 account is created

if (iCount > 0)

{

cout << "\n\n";

}

oReceptionist[iCount].GetDetails(iCount);

}

// Formatting

cout << "\n\n";

// Check login details against registered receptionist accounts to see if the details are correct or incorrect

do

{

for (\_\_int16 iCount = 0; iCount < oReceptionist.size(); ++iCount)

{

if (bCheck == false) // If the details are incorrect

{

oSystem.LoginPromp(); // Prompts user to login

cin >> sUsername; // Get username input

sPassword = HidePasswordDetails(); // Get password input

cout << '\n'; // Formatting

for (\_\_int16 iCount = 0; iCount < oReceptionist.size(); ++iCount)

{

bCheck = oReceptionist[iCount].Login(sUsername, sPassword); // Checks if login details are correct or incorrect

if (bCheck == true)

{

break; // Breaks out of the loop if the username and password details passed match a receptionist accounts details

}

}

if (bCheck == false && iFail < 3) // if the username and password details were incorrect and the account hasn't been locked out

{

oReceptionist[0].IncorrectDetails(); // Displays incorrect details message

}

else

{

oReceptionist[0].LoginConfirmation(bCheck); // Logs the user into the system if details are correct, locks out the user and exists program if incorrect

// Lockout the user from the program

if (bCheck == false)

{

system("PAUSE"); // Pauses the program so the user can see the lockout message

return 0; // Ends the program

}

}

}

iFail++; // If details are incorrect increment iFail, used for lockout

}

} while (iFail <= 3 && bCheck == false); // Allows 3 attempts to login

// RECEPTIONIST SYSTEM CONFIGURATION MENU AND FUNCTIONALITY

do

{

// Receptionist logged into system

do

{

// Setup menu for configuring the system

oSystem.SetupMenu();

// Get users menu option choice

cin >> iChoice;

// Error checking to stop user from entering bad input such as text or special characters

iChoice = OnlyIntegerErrorChecking(iChoice);

// Error checking to stop user from selecting an option not on the menu

iChoice = MenuOptionErrorChecking("SetupMenu", iChoice);

// Error checking to stop user from selecting same option twice

bCheck = oSystem.PreviousChoiceErrorChecking(iChoice, iPreviousChoice, bCheck);

} while (bCheck == false); // Loops until valid input choice has been entered

// Assigns previous choice number to variable, used for PreviousChoiceErrorChecking, making sure the user doesn't select the same option twice

iPreviousChoice = iChoice;

switch (iChoice)

{

case 1:

// Add doctor functionality

// Calls and assigns number of doctors

iNumberOfDoctors = oDoctor[0].GetNumberOfObjects("doctor");

// Create vector of doctors to inputted amount

oDoctor.resize(iNumberOfDoctors);

// Get number of doctor accounts, enter username, password and specialist area for each

for (\_\_int16 iCount = 0; iCount < iNumberOfDoctors; ++iCount)

{

if (iCount == 0)

{

iPreviousDoctorID = 0; // Sets first doctor previous ID to 0

}

else

{

iPreviousDoctorID = oDoctor[iCount - 1].ReturnID(); // Gets previous doctor objects ID and stores it in iPreviousDoctorID

}

oDoctor[iCount].GenerateID(iPreviousDoctorID); // Generates new ID for current doctor object based off the previous doctor object ID

oDoctor[iCount].GetDetails(iCount); // Gets username and password information for doctor object

oDoctor[iCount].GetDoctorDetails(iCount); // Gets specialist area information for doctor object

}

// Increment iMenuCount to stay in menu loop/break out when all configuration information has been inputted

++iMenuCount;

// Sends last menu option, removes last menu option from the next menu loop

oSystem.LastMenuOption("DOCTOR");

break;

case 2:

// Add surgery information functionality

// Calls and assigns number of surgery's

iNumberOfSurgerys = oSurgery[0].GetNumberOfObjects("surgery");

// Create vector of surgerys to inputted amount

oSurgery.resize(iNumberOfSurgerys);

// Get number of surgerys to create, enter surgery name and the name of the doctor who does the surgery

for (\_\_int16 iCount = 0; iCount < iNumberOfSurgerys; ++iCount)

{

oSurgery[iCount].GetSurgeryDetails(iCount);

}

// Increment iMenuCount to stay in menu loop/break out when all configuration information has been inputted

++iMenuCount;

// Sends last menu option, removes last menu option from the next menu loop

oSystem.LastMenuOption("SURGERY");

break;

case 3:

// Exit program functionality

return 0;

break;

}

} while (iMenuCount < 2); // Breaks out when both doctor and surgery information has been completed

// RECEPTIONIST LOGGED IN MENU AND FUNCTIONALITY

do

{

// System has been configured with information

// Outputs receptionist menu for booking and adding new accounts

oSystem.LoggedInMenu();

// Get users menu option choice

cin >> iChoice;

// Error checking to stop user from entering bad input such as text or special characters

iChoice = OnlyIntegerErrorChecking(iChoice);

// Error checking to stop user from selecting an option not on the menu

iChoice = MenuOptionErrorChecking("LoggedInMenu", iChoice);

// Get receptionist menu choice

switch (iChoice)

{

case 1:

// BOOK CUSTOMER FUNCTIONAILITY

// New customer

// Adding customer information

cout << "\nAdd customer information\n";

// CUSTOMER FUNCTIONS

// Get previous customer object id and store in iPreviousCustomerID, first time is 0

iPreviousCustomerID = oNewCustomer.ReturnID();

// Pass iPreviousID and increment for new customer object id

oNewCustomer.GenerateID(iPreviousCustomerID);

// Get customer name and password details

oNewCustomer.GetDetails(iPreviousCustomerID);

// Formatting

cout << '\n';

// Get customer phone number, address and ailment details

oNewCustomer.GetCustomerDetails(iPreviousCustomerID);

// BOOKING FUNCTIONS

// Get previous booking object id and store in iPreviousBookingID, first time is 0

iPreviousBookingID = oNewBooking.ReturnID();

// Pass iPreviousBookingID and increment for new booking object id

oNewBooking.GenerateID(iPreviousBookingID);

// Add oNewCustomer to the back of the oCustomer vector

oCustomer.push\_back(oNewCustomer);

// Book customer appointment surgery details

// Error checking to check whether surgery has available slot

do

{

// Outputs customers ailment

oNewCustomer.ShowAilment(iPreviousCustomerID);

// Prompts user to select a surgery from the options

oSystem.SelectSurgeryPrompt();

// Outputs list of surgeries with their details for receptionsit to choose from to assign to customer

for (\_\_int16 iCount = 0; iCount < oSurgery.size(); ++iCount)

{

oSurgery[iCount].ShowSurgeryDetails(iCount);

}

// Get surgery number

cin >> iSurgeryChoice;

// Error checking to stop user from entering bad input such as text or special characters

iSurgeryChoice = OnlyIntegerErrorChecking(iSurgeryChoice);

// Stores size of oSurgery vector for menu error checking

iSurgeryObjectSize = oSurgery.size();

// Error checking to stop user from selecting an option that doesn't exist

iSurgeryChoice = OptionErrorChecking(iSurgeryChoice, iSurgeryObjectSize);

// Minus 1 from the choice to get correct index of surgery vector

iSurgeryChoice = iSurgeryChoice - 1;

// Does the surgery have an available slot

bCheck = oSurgery[iSurgeryChoice].NoAvailableSlots(bCheck);

} while (bCheck == false); // Loops until the surgery chosen is one with available slots

// Error checking to check whether doctor has available slot

do

{

// Formatting

cout << '\n';

// Book customer appointment with doctor

// Outputs customers ailment

oNewCustomer.ShowAilment(iPreviousCustomerID);

// Prompts user to select doctor from the options

oSystem.SelectDoctorPrompt();

// Outputs list of doctors with their details for receptionsit to choose from to assign to customer

for (\_\_int16 iCount = 0; iCount < oDoctor.size(); ++iCount)

{

oDoctor[iCount].ShowDoctorDetails(iCount);

}

// Get doctor number

cin >> iDoctorChoice;

// Error checking to stop user from entering bad input such as text or special characters

iDoctorChoice = OnlyIntegerErrorChecking(iDoctorChoice);

// Stores size of oDoctor vector for menu error checking

iDoctorObjectSize = oDoctor.size();

// Error checking to stop user from selecting an option that doesn't exist

iDoctorChoice = OptionErrorChecking(iDoctorChoice, iDoctorObjectSize);

// Minus 1 from the choice to get correct index of doctor vector

iDoctorChoice = iDoctorChoice - 1;

// Does the doctor have an available slot

bCheck = oDoctor[iDoctorChoice].NoAvailableSlots(bCheck);

} while (bCheck == false); // Loops until the doctor chosen is one with available slots

// Get doctor ID for booking details

iDoctorID = oDoctor[iDoctorChoice].ReturnID();

// Get doctor name for booking details

sDoctorName = oDoctor[iDoctorChoice].ReturnName();

// Returns the name of the customer object back to the main

sCustomerName = oCustomer[iPreviousCustomerID].ReturnName();

// Returns customer ailment back to main for booking details

sCustomerAilment = oCustomer[iPreviousCustomerID].ReturnAilment();

// BOOKING TIME FUNCTIONS

// Get the local computer time

tNow = time(0);

// Convert the time to string for a booking date

sBookingDate = ctime(&tNow);

// Removes new line '\n' from the string

sBookingDate.erase(remove(sBookingDate.begin(), sBookingDate.end(), '\n'), sBookingDate.end());

// Gets booking details, stores them in the NewBooking object. Passes doctor choice (ID), doctor name and customer name to be stored

oNewBooking.GetBookingDetails(iDoctorID, iSurgeryChoice, sDoctorName, sCustomerName, sCustomerAilment, sBookingDate);

// Formatting

cout << "\n\n";

// Get booking confirmation of fee

bCheck = oNewBooking.BookingCostConfirmation(bCheck, iChoice);

// If denied or accepted

if (bCheck == false)

{

// Denied booking confirmation

cout << "You denied the booking confirmation - booking not created\n";

// Exit booking

break;

}

else

{

// Accepted booking confirmation

cout << "You accepted the booking confirmation - booking created\n";

// Add oNewBooking to the back of the oBooking vector

oBooking.push\_back(oNewBooking);

// Removes appointment slot for doctor when a booking is created with the doctor

oDoctor[iDoctorChoice].MinusAppointmentSlot();

// Removes available slot for surgery wehn a booking is created

oSurgery[iSurgeryChoice].MinusAvailableSlot();

}

break;

case 2:

// Display all bookings in the system

// Formatting

cout << "\n\n";

// Displays all bookings in the system

for (int iCount = 0; iCount < oBooking.size(); ++iCount)

{

oBooking[iCount].ReviewAllBookings();

oSurgery[iSurgeryChoice].ShowSurgeryBookedDetails();

}

break;

case 3:

// View all doctors in system

// Formatting

cout << "\n\n";

// Loops and prints doctor details

for (\_\_int16 iCount = 0; iCount < oDoctor.size(); ++iCount)

{

oDoctor[iCount].ShowDoctorDetails(iCount);

}

break;

case 4:

// View all customers in system

// Formatting

cout << "\n\n";

// Loops and prints customer details

for (\_\_int16 iCount = 0; iCount < oCustomer.size(); ++iCount)

{

oCustomer[iCount].ShowCustomerDetails(iCount);

}

break;

case 5:

// View all surgeries in system

// Formatting

cout << "\n\n";

// Loops and prints surgery details

for (\_\_int16 iCount = 0; iCount < oSurgery.size(); ++iCount)

{

oSurgery[iCount].ShowSurgeryDetails(iCount);

}

break;

case 6:

// View all receptionists in system

// Formatting

cout << "\n\n";

// Loops and prints receptionist details

for (\_\_int16 iCount = 0; iCount < oReceptionist.size(); ++iCount)

{

oReceptionist[iCount].DisplayDetails(iCount);

}

break;

case 7:

// Add more doctor accounts

cout << "\n\nAdd a new doctor\n\n";

// Gets the current size of the oDoctor vector

iDoctorObjectSize = oDoctor.size();

// Gets the previous doctor ID, returns and assigns

iPreviousDoctorID = oDoctor[iDoctorObjectSize - 1].ReturnID();

// Generates new ID by incrementing the previous doctor object ID

oNewDoctor.GenerateID(iPreviousDoctorID);

// Gets the name and password information for the new doctor account

oNewDoctor.GetDetails(iPreviousDoctorID);

// Gets the specialist area information for hte new doctor account

oNewDoctor.GetDoctorDetails(iPreviousDoctorID);

// Adds the oNewDoctor object to the back of the vector, adding a new doctor account

oDoctor.push\_back(oNewDoctor);

cout << "Doctor account added successfully\n\n";

break;

case 8:

// Add more receptionist accounts

cout << "\n\nAdd a new receptionist\n\n";

// Gets the current size of the oReceptionist vector

iReceptionistObjectSize = oReceptionist.size();

// Get name and password information for the new receptionist account

oNewReceptionist.GetDetails(iReceptionistObjectSize);

// Adds the oNewReceptionist object to the back of the vector, adding a new receptionist account

oReceptionist.push\_back(oNewReceptionist);

cout << "\nReceptionist account added successfully\n\n";

break;

case 9:

// Add more surgery accounts

cout << "\n\nAdd a new surgery\n\n";

// Gets the current size of the oReceptionist vector

iSurgeryObjectSize = oSurgery.size();

// Gets surgery information

oNewSurgery.GetSurgeryDetails(iSurgeryObjectSize);

// Adds oNewSurgery object to the back of the vector, adding a new surgery account

oSurgery.push\_back(oNewSurgery);

cout << "\nSurgery account added successfully\n\n";

break;

case 10:

// Delete last booking created in system

// Get size of booking vector

iBookingObjectSize = oBooking.size();

// Negates error checking to allow user to delete all bookings in the system

iObjectSize = 2;

// Gets confirmation from the user before deleting

bCheck = DeleteLastObjectConfirmation(bCheck, "booking", iObjectSize);

// Get last booking in the system doctor ID

iBookingDoctorID = oBooking[iBookingObjectSize - 1].ReturnDoctorID();

if (bCheck == true)

{

// Find doctor tied to booking, add 1 back to available slots

oDoctor[iBookingDoctorID - 1].AddAppointmentSlot();

DeleteLastObject(&oBooking);

cout << "\nSuccessfully deleted last booking\n\n";

}

else

{

cout << "\nNo bookings deleted\n\n";

}

break;

case 11:

// Delete last doctor created in system

// Object size for error checking to make sure at least 1 object remains in the system and prevents deletion if only 1

iObjectSize = oDoctor.size();

// Gets confirmation from the user before deleting

bCheck = DeleteLastObjectConfirmation(bCheck, "doctor", iObjectSize);

if (bCheck == true)

{

DeleteLastObject(&oDoctor);

cout << "\nSuccessfully deleted last doctor account\n\n";

}

else

{

cout << "\nNo doctors deleted\n\n";

}

break;

case 12:

// Delete last surgery created in system

// Object size for error checking to make sure at least 1 object remains in the system and prevents deletion if only 1

iObjectSize = oSurgery.size();

// Gets confirmation from the user before deleting

bCheck = DeleteLastObjectConfirmation(bCheck, "surgery", iObjectSize);

if (bCheck == true)

{

DeleteLastObject(&oSurgery);

cout << "\nSuccessfully deleted last surgery account\n\n";

}

else

{

cout << "\nNo surgeries deleted\n\n";

}

break;

case 13:

// Delete last receptionist created in system

// Object size for error checking to make sure at least 1 object remains in the system and prevents deletion if only 1

iObjectSize = oReceptionist.size();

// Gets confirmation from the user before deleting

bCheck = DeleteLastObjectConfirmation(bCheck, "receptionist", iObjectSize);

if (bCheck == true)

{

DeleteLastObject(&oReceptionist);

cout << "\nSuccessfully deleted last receptionist account\n\n";

}

else

{

cout << "\nNo receptionists deleted\n\n";

}

break;

case 14:

// Exit program

// End of program

cout << "Thank you for using the doctor's surgery program\n";

return 0;

break;

}

} while (iChoice != 14); // Loops until the user selects option 6 'exit'

\_getch();

return 0;

}