#include <iostream>

#include <cstdlib>

#include <fstream>

#include <string>

#include <iomanip>

#define U\_MEMBERS 7

#define P\_MEMBERS 11

#define NOT\_CALCULATED -2

using namespace std;

// auxiliary\_functions.h

string unparseRecord(string record[],int size)

{

// Converts all the attributes into a comma separated

// string equivalent to a row in CSV file

string row;

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

{

// cout<<"inside loop"<<endl;

row += record[i];

// strcat(row,record[i].c\_str());

if(i == size - 1)

row += "\n";

// strcat(row,"\n");

else

row += ",";

// strcat(row,",");

// return the row

}

return row;

}

bool insensitiveCompare(string s1,string s2)

{

if(s1.length() == s2.length())

{

int length = s1.length();

int i = 0;

while(i < length)

{

if(toupper(s1[i]) != toupper(s2[i]))

return false;

i++;

}

return true;

}

else

return false;

}

bool isComma(char c)

{

if(c == ',')

return true;

else

return false;

}

bool isNewLine(char c)

{

if(c == '\n')

return true;

else

return false;

}

void endString(char str[],int index)

{

str[index] = '\0';

}

void truncate(int &x)

{

x = 0;

}

void skipLine(std::ifstream &f1)

{

while(f1.get() != '\n');

}

void getLine(std::ifstream &f1,string &str)

{

int i = 0;

char c;

while(!isNewLine(c = f1.get()))

str += c;

str += '\n';

}

void putLine(std::ofstream &f2,string str)

{

int i = 0;

while(!isNewLine(str[i]))

f2.put(str[i++]);

f2.put('\n');

}

void skipComma(std::ifstream &f1)

{

while(!isComma(f1.get()));

}

std::string\*\* dynamic2DString(int row,int column)

{

std::string\*\* array = new std::string\*[row];

int i;

for(i = 0;i < column;i++)

array[i] = new std::string[column];

return array;

}

// menu.h

void ignore()

{

string sth;

getline(cin,sth,'\n');

}

void main\_menu()

{

cout<<"\t\t\t\t\t\tWelcome to real estate database manager"<<endl;

cout<<"\t\t\t\t\t\t---------------------------------------"<<endl<<endl;

cout<<"\t\t\t\t\t\t1.Admin login/register"<<endl; // UNique code required to register as admin

cout<<"\t\t\t\t\t\t2.User login/register"<<endl; //

cout<<"\t\t\t\t\t\t3.Guest Mode"<<endl;

cout<<"\t\t\t\t\t\t4.Exit"<<endl;

}

void admin\_menu()

{

cout<<"\t\t\t\t\t\tAdmin Menu"<<endl;

cout<<"\t\t\t\t\t\t----------"<<endl<<endl;

cout<<"\t\t\t\t\t\t1.Login"<<endl;

cout<<"\t\t\t\t\t\t2.Return to main menu"<<endl;

cout<<"\t\t\t\t\t\t3.Exit"<<endl;

}

void user\_menu()

{

cout<<"\t\t\t\t\t\tUser Menu"<<endl;

cout<<"\t\t\t\t\t\t----------"<<endl<<endl;

cout<<"\t\t\t\t\t\t1.Login"<<endl;

cout<<"\t\t\t\t\t\t2.Register"<<endl;

cout<<"\t\t\t\t\t\t3.Forgot Password"<<endl;

cout<<"\t\t\t\t\t\t4.Return to main menu"<<endl;

cout<<"\t\t\t\t\t\t5.Exit"<<endl;

}

void guest\_menu()

{

cout<<"\t\t\t\t\t\tGuest Menu"<<endl;

cout<<"\t\t\t\t\t\t----------"<<endl<<endl;

cout<<"\t\t\t\t\t\t1.Search Properties"<<endl;

cout<<"\t\t\t\t\t\t2.Return to main menu"<<endl;

cout<<"\t\t\t\t\t\t3.Exit"<<endl;

}

void admin\_loggedin\_menu()

{

cout<<"\t\t\t\t\t\tAdmin Interface"<<endl;

cout<<"\t\t\t\t\t\t---------------"<<endl<<endl;

cout<<"\t\t\t\t\t\t1.Search Properties"<<endl;

cout<<"\t\t\t\t\t\t2.Add a Property"<<endl;

cout<<"\t\t\t\t\t\t3.Update a Property"<<endl;

cout<<"\t\t\t\t\t\t4.Delete a property"<<endl;

cout<<"\t\t\t\t\t\t5.See pending registrations"<<endl;

cout<<"\t\t\t\t\t\t6.Logout"<<endl;

}

void user\_loggedin\_menu()

{

cout<<"\t\t\t\t\t\tUser's Interface"<<endl;

cout<<"\t\t\t\t\t\t----------------"<<endl<<endl;

cout<<"\t\t\t\t\t\t1.Search properties"<<endl;

cout<<"\t\t\t\t\t\t2.My owned properties"<<endl;

cout<<"\t\t\t\t\t\t3.Register for property"<<endl;

cout<<"\t\t\t\t\t\t4.Check Registration status"<<endl;

cout<<"\t\t\t\t\t\t6.Update profile"<<endl;

cout<<"\t\t\t\t\t\t9.Logout"<<endl;

}

// CSVProcessor.h

class CSVProcessor

{

private:

std::string filehandle;

// int records;

int noOfAttributes;

int noOfRecords;

public:

CSVProcessor(){

// filehandle = NULL;

noOfAttributes = NOT\_CALCULATED;

noOfRecords = NOT\_CALCULATED;

}

CSVProcessor(std::string filename)

{

this->filehandle = filename + ".csv";

noOfAttributes = NOT\_CALCULATED;

noOfRecords = NOT\_CALCULATED;

/\*everytime check if the mode allows the function to be executed or not

i.e, if user wants to add or delete record, check if mode is write or append, otherwise return false

\*/

}

int countRecords()

{

if(noOfRecords == NOT\_CALCULATED)

{

noOfRecords = 0;

std::ifstream f1(filehandle.c\_str(),std::ios::in);

while(!f1.eof())

{

if(f1.get() == '\n')

noOfRecords++;

}

noOfRecords--;

f1.close();

}

return noOfRecords;

}

int countAttributes()

{

if(noOfAttributes == NOT\_CALCULATED)

{

noOfAttributes = 0;

std::ifstream f1(filehandle.c\_str(),std::ios::in);

char c;

do{

c = f1.get();

if(c == '"')

{

while(f1.get() != '"');

c = f1.get();

}

if(c == ',')

noOfAttributes++;

}while(c != '\n');

noOfAttributes++;

}

return noOfAttributes;

}

std::string\*\* getRecords()

{

std::ifstream f1(filehandle.c\_str(),std::ios::in);

std::string\*\* records = dynamic2DString(countRecords(),countAttributes());

int recordsIndex = 0;

skipLine(f1);

while(f1.peek() != EOF)

// while(!f1.eof())

{

// if(f1.peek() == EOF)

// break;

string row;

getLine(f1,row);

records[recordsIndex++] = parseRecord(row);

}

f1.close();

return records;

}

std::string\* parseRecord(string row)

{

// takes the string containing commas and returns the array of strings(separate attributes in separate strings)

std::string\* record = new std::string[countAttributes()];

string tempString;

int rowIndex ,recordIndex,tempIndex;

rowIndex = recordIndex = tempIndex = 0;

while(!isNewLine(row[rowIndex])) // till the end of string

{

if(row[rowIndex] == '"')

{

do

{

tempString += row[rowIndex++];

}while(row[rowIndex] != '"');

tempString += '"';

}

else if(isComma(row[rowIndex]))

{

// endString(tempString,tempIndex);

record[recordIndex++] = tempString;

// truncate(tempIndex);

tempString.clear();

}

else

{

tempString += row[rowIndex];

}

rowIndex++;

}

// endString(tempString,tempIndex);

record[recordIndex]= tempString;

return record;

}

std::string\* getRecord(std::string recordId)

{

// Used for searching purposes... searching a particular record.

// Takes the unique record id as argument and returns the record(array of strings)

std::ifstream f1(filehandle.c\_str(),std::ios::in);

std::string \*record = new std::string[countAttributes()];

skipLine(f1);

bool found = false;

while(f1.peek() != EOF)

// while(!f1.eof())

{

// if(f1.peek() == EOF)

// break;

string row;

getLine(f1,row);

record = parseRecord(row);

if(record[0] == recordId)

{

found = true;

break;

}

}

if(found)

return record;

else

return NULL;

}

std::string\* getAttributes()

{

//

std::ifstream f1(filehandle.c\_str(),std::ios::in);

std::string \*attributes = new std::string[countAttributes()];

string row;

getLine(f1,row);

attributes = parseRecord(row);

return attributes;

}

bool addRecord(string row)

{

std::ofstream f1(filehandle.c\_str(),ios::out|ios::app);

f1.seekp(ios::end);

putLine(f1,row);

f1.close();

return true;

}

bool deleteRecord(std::string recordID)

{

// takes the unique record Id and deletes the corresponding record from the database

std::ifstream f1(filehandle.c\_str(),std::ios::in);

std::ofstream f2("temp.csv",std::ios::out);

string row;

getLine(f1,row);

putLine(f2,row);

bool found = false;

std::string \*record = new std::string[countAttributes()];

while(f1.peek() != EOF)

// while(!f1.eof())

{

// if(f1.peek() == EOF)

// break;

getLine(f1,row);

record = parseRecord(row);

if(record[0] == recordID)

{

found = true;

continue;

}

putLine(f2,row);

}

f1.close();

f2.close();

remove(filehandle.c\_str());

rename("temp.csv",filehandle.c\_str());

return found;

}

bool updateRecord(std::string recordID,string newRow)

{

// Take argument the id(primary key) of the record to be modified and the new row to be

std::ifstream f1(filehandle.c\_str(),std::ios::in);

std::ofstream f2("temp.csv",std::ios::out);

string row;

getLine(f1,row);

putLine(f2,row);

row.clear();

bool found = false;

std::string \*record = new std::string[countAttributes()];

while(f1.peek() != EOF)

// while(!f1.eof())

{

// if(f1.peek() == EOF)

// break;

getLine(f1,row);

record = parseRecord(row);

if(record[0] == recordID)

{

found = true;

putLine(f2,newRow);

row.clear();

continue;

}

putLine(f2,row);

row.clear();

}

f1.close();

f2.close();

remove(filehandle.c\_str());

rename("temp.csv",filehandle.c\_str());

return found;

}

};

// user.h

class base{

protected:

string id;

string name;

public:

bool isValid;

};

// property.h

class Property : public base

{

private:

string options[11];

string location;

string locality;

string type;

string size;

string quantity\_left;

string price;

string construction\_status;

string description;

string contact\_details;

public:

Property()

{

options[0] = "id";

options[1] = "name";

options[2] = "location";

options[3] = "locality";

options[4] = "type";

options[5] = "size";

options[6] = "quantity\_left";

options[7] = "price";

options[8] = "construction\_status";

options[9] = "description";

options[10] = "contact\_details";

isValid = false;

}

Property(string id)

{

options[0] = "id";

options[1] = "name";

options[2] = "location";

options[3] = "locality";

options[4] = "type";

options[5] = "size";

options[6] = "quantity\_left";

options[7] = "price";

options[8] = "construction\_status";

options[9] = "description";

options[10] = "contact\_details";

CSVProcessor p("property");

string \*record = p.getRecord(id);

if(record != NULL)

{

isValid = true;

(id = record[0]);

(name = record[1]);

(location = record[2]);

(locality = record[3]);

(type = record[4]);

(size = record[5]);

(quantity\_left = record[6]);

(price = record[7]);

(construction\_status = record[8]);

(description = record[9]);

(contact\_details = record[10]);

}

}

Property(string \*record)

{

options[0] = "id";

options[1] = "name";

options[2] = "location";

options[3] = "locality";

options[4] = "type";

options[5] = "size";

options[6] = "quantity\_left";

options[7] = "price";

options[8] = "construction\_status";

options[9] = "description";

options[10] = "contact\_details";

isValid = true;

(id = record[0]);

(name = record[1]);

(location = record[2]);

(locality = record[3]);

(type = record[4]);

(size = record[5]);

(quantity\_left = record[6]);

(price = record[7]);

(construction\_status = record[8]);

(description = record[9]);

(contact\_details = record[10]);

}

static void showMinimalAttributes()

{

cout<<left;

cout<<setw(15)<<" ";

cout<<setw(15)<<"PID";

cout<<setw(15)<<"PName";

cout<<setw(15)<<"Location";

cout<<setw(15)<<"Locality";

cout<<setw(15)<<"Type";

cout<<setw(15)<<"Size";

cout<<setw(15)<<"quantity\_left";

cout<<setw(15)<<"Construction Status"<<endl<<endl;

}

void getData()

{

CSVProcessor p("property");

string \*\*properties = p.getRecords();

int noOfProperties = p.countRecords();

for(int i = 1;i < 11; i++)

{

// cout<<options[i]<<endl;

cout<<"Enter "<<options[i]<<" : ";

getline(cin,(\*this)[options[i]],'\n');

}

(\*this)[options[0]] = to\_string(stoi(properties[noOfProperties-1][0]) + 1);

cout<<"Unique property\_id generated is : "<<(\*this)[options[0]]<<endl;

cout<<"Note down and press ENTER"<<endl;

ignore();

}

void addProperty()

{

string row;

for(int i = 0;i < 11; i++)

{

if(i == 9)

row += "\"";

row += (\*this)[options[i]];

if(i == 9)

row += "\"";

if(i == 10)

row += "\n";

else

row += ",";

}

CSVProcessor p("property");

p.addRecord(row);

}

void deleteProperty()

{

CSVProcessor p("property");

p.deleteRecord(id);

}

void updateProperty()

{

CSVProcessor p("property");

string row;

for(int i = 0;i < 11; i++)

{

string temp;

cout<<"Enter new "<<options[i]<<" (leave empty if unchanged) : ";

getline(cin,temp,'\n');

if(temp.length() == 0)

row += (\*this)[options[i]];

else

row += temp;

if(i == 10)

row += "\n";

else

row += ",";

}

// cout<<"row = "<<row<<endl;

p.updateRecord(id,row);

}

void detailDisplay()

{

for(int i = 0;i < P\_MEMBERS; i++)

cout<<options[i]<<" : "<<(\*this)[options[i]]<<endl;

}

void displayMinimal()

{

// displays the minimal details when someone searches

cout<<setw(15)<<this->location;

cout<<setw(15)<<this->locality;

cout<<setw(15)<<this->type;

cout<<setw(15)<<this->size;

cout<<setw(15)<<this->price;

cout<<setw(15)<<this->construction\_status<<endl;

}

/\* int getProperties()

{

// use this pointer

}\*/

string & operator[](string index)

{

try

{

if(index == options[0])

return id;

else if(index == options[1])

return name;

else if(index == options[2])

return location;

else if(index == options[3])

return locality;

else if(index == options[4])

return type;

else if(index == options[5])

return size;

else if(index == options[6])

return quantity\_left;

else if(index == options[7])

return price;

else if(index == options[8])

return construction\_status;

else if(index == options[9])

return description;

else if(index == options[10])

return contact\_details;

else

throw 100;

}

catch(int i)

{

cout<<"ERROR : Invalid index !!!"<<endl;

}

}

};

class User : base

{

string options[U\_MEMBERS];

string username;

string password;

string phone\_no;

string e\_mail;

string properties;

public:

User()

{

isValid = false;

options[0] = "id";

options[1] = "username";

options[2] = "password";

options[3] = "name";

options[4] = "phone\_no";

options[5] = "e\_mail";

options[6] = "properties";

properties = "0";

}

User(string id)

{

options[0] = "id";

options[1] = "username";

options[2] = "password";

options[3] = "name";

options[4] = "phone\_no";

options[5] = "e\_mail";

options[6] = "properties";

CSVProcessor p("user");

string \*record = p.getRecord(id);

if(record != NULL)

{

isValid = true;

id = record[0];

username = record[1];

password = record[2];

name = record[3];

phone\_no = record[4];

e\_mail = record[5];

properties = record[6];

}

}

User(string \*record)

{

options[0] = "id";

options[1] = "username";

options[2] = "password";

options[3] = "name";

options[4] = "phone\_no";

options[5] = "e\_mail";

options[6] = "properties";

isValid = true;

// cout<<"inside constructor"<<endl;

// for(int i = 0;i < 6; i++)

// cout<<record[i]<<endl;

id = record[0];

username = record[1];

password = record[2];

name = record[3];

phone\_no = record[4];

e\_mail = record[5];

// for(int i = 0;i < 6; i++)

// {

// cout<<options[i]<<" : "<<(\*this)[options[i]]<<endl;

// }

}

void initialise(string \*record)

{

// cout<<"inside constructor"<<endl;

// for(int i = 0;i < 6; i++)

// cout<<record[i]<<endl;

id = record[0];

username = record[1];

password = record[2];

name = record[3];

phone\_no = record[4];

e\_mail = record[5];

properties = record[6];

// for(int i = 0;i < 6; i++)

// {

// cout<<options[i]<<" : "<<(\*this)[options[i]]<<endl;

// }

}

void myProperties()

{

system("cls");

CSVProcessor p(username);

cout<<setw(25)<<" ";

int noOfProperties = p.countRecords();

if(noOfProperties == 0)

{

cout<<endl<<endl<<endl;

cout<<setw(25)<<"No properties bought by the user"<<endl;

}

else

{

string \*\*property = p.getRecords();

cout<<".."<<endl;

Property::showMinimalAttributes();

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

{

cout<<setw(25)<<" ";

cout<<setw(25)<<property[i][0];

for(int j = 2;j <= 7; j++)

cout<<setw(25)<<property[i][j];

cout<<endl;

}

}

}

void createFile()

{

CSVProcessor p1("property");

CSVProcessor p2(username.c\_str());

string \*attributes = p1.getAttributes();

p2.addRecord(unparseRecord(attributes,11));

}

bool deleteUser()

{

CSVProcessor p("user");

p.deleteRecord(id);

}

void getData()

{

CSVProcessor p("user");

string \*\*users = p.getRecords();

int noOfUsers = p.countRecords();

for(int i = 1;i <= U\_MEMBERS - 2; i++)

{

XYZ:

string temp;

cout<<"Enter "<<options[i]<<" : ";

getline(cin,temp,'\n');

if(i == 1 || i == 4 || i == 5)

{

for(int k = 0;k < noOfUsers; k++)

{

if(users[k][i] == temp)

{

cout<<options[i]<<" already in use...press enter to try again"<<endl;

ignore();

goto XYZ;

}

}

}

(\*this)[options[i]] = temp;

}

if(noOfUsers == 0)

(\*this)[options[0]] = "usr10001";

else

{

string x = users[noOfUsers-1][0].substr(3);

(\*this)[options[0]] = "usr" + to\_string(stoi(x) + 1);

}

(\*this)[options[6]] = "0";

cout<<"Unique user\_id generated is : "<<(\*this)[options[0]]<<endl;

cout<<"Note down and press ENTER"<<endl;

ignore();

}

void addUser()

{

CSVProcessor p("user");

string row;

for(int i = 0;i < U\_MEMBERS; i++)

{

row += (\*this)[options[i]];

if(i == U\_MEMBERS - 1)

row += "\n";

else

row += ",";

}

p.addRecord(row);

}

void updateProfile()

{

CSVProcessor p("user");

string row;

/\* for(int i = 0;i < 6; i++)

{

cout<<options[i]<<" : "<<(\*this)[options[i]]<<endl;

}\*/

for(int i = 0;i < U\_MEMBERS; i++)

{

string temp;

cout<<"Enter new "<<options[i]<<" (leave empty if unchanged) : ";

getline(cin,temp,'\n');

if(temp.length() == 0)

row += (\*this)[options[i]];

else

row += temp;

if(i == U\_MEMBERS - 1)

row += "\n";

else

row += ",";

}

// cout<<"row = "<<row<<endl;

p.updateRecord(id,row);

}

string & operator[](string index)

{

try

{

if(index == options[0])

return id;

else if(index == options[1])

return username;

else if(index == options[2])

return password;

else if(index == options[3])

return name;

else if(index == options[4])

return phone\_no;

else if(index == options[5])

return e\_mail;

else if(index == options[6])

return properties;

else

throw 100;

}

catch(int i)

{

cout<<"ERROR : Invalid index !!!"<<endl;

}

}

};

User u1;

namespace stringNumber

{

string extractNumber(string str)

{

string temp;

int i = 0;

while(isdigit(str[i]) || str[i] == '.')

temp += str[i++];

return temp;

}

bool compare(string budget,string cost)

{

string a = extractNumber(budget);

string b = extractNumber(cost);

if(atof(a.c\_str()) < atof(b.c\_str()))

return false;

else

return true;

}

}

void getProperties(string parameters[])

{

cout<<"Press enter to show matching results..."<<endl;

ignore();

system("cls");

Property::showMinimalAttributes();

CSVProcessor par("property");

int noOfProperties = par.countRecords();

// cout<<"Number of records = "<<noOfProperties<<endl;

string \*\*records = par.getRecords();

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

{

bool status = true;

int k = 0;

for(int j = 2;j <= 4; j++,k++)

{

if(parameters[k].length() > 0)

{

if(!insensitiveCompare(parameters[k],records[i][j]))

{

status = false;

break;

}

}

}

if(parameters[3].length() > 0)

{

if(!stringNumber::compare(parameters[3],records[i][7]))

{

status = false;

}

}

if(status == true)

{

cout<<setw(15)<<" ";

for(int j = 0;j <= 7; j++)

cout<<setw(15)<<records[i][j];

cout<<endl;

}

}

}

void search\_properties();

void ignore();

void admin\_loggedin\_menu();

void admin\_menu();

void user\_loggedin\_menu();

void user\_menu();

void getProperties(string []);

void guest\_menu();

void add\_property();

void delete\_property();

void update\_property();

void purchased\_properties();

void register\_property();

void update\_profile();

void forgot\_password();

void pending\_registrations();

void admin\_interface()

{

while(1)

{

E:

system("clear");

admin\_loggedin\_menu();

int choice;

while(!(cin>>choice))

{

cout<<"Enter integers only"<<endl;

cin.clear();

cin.ignore(10000,'\n');

}

cin.ignore(10000,'\n');

switch(choice)

{

case 1:

search\_properties();

break;

case 2:

add\_property();

// add a property

break;

case 3:

// update a property

update\_property();

break;

case 4:

delete\_property();

break;

case 5:

pending\_registrations();

break;

// see pending registrations

case 6:

return;// logout

default:

cout<<"invalid choice!!!"<<endl;

cout<<"Press enter to continue..."<<endl;

ignore();

goto E;

}

}

}

void user\_interface()

{

while(1)

{

F:

system("cls");

user\_loggedin\_menu();

int choice;

while(!(cin>>choice))

{

cout<<"Enter integers only"<<endl;

cin.clear();

cin.ignore(10000,'\n');

}

cin.ignore(10000,'\n');

switch(choice)

{

case 1:

search\_properties();

break;// see properties

case 2:

cout<<"Press enter to continue...";

ignore();

purchased\_properties();

break;

case 3:

register\_property();

break;

// purchase a prop

case 4:

update\_profile();// update profile

break;

case 5:

cout<<"logout successful.."<<endl;

cout<<"press ENTER to continue..."<<endl;// logout

ignore();

return;

default:

cout<<"invalid choice!!!"<<endl;

cout<<"Press enter to continue..."<<endl;

ignore();

goto F;

}

}

}

void admin\_login()

{

// Take username and password of admin and cross check it

// from the admin.csv file.

CSVProcessor p("admin");

string\*\* admins = p.getRecords();

int noOfAdmins = p.countRecords();

ALOG:

cout<<"Enter the admin-id : ";

string id;

getline(cin,id,'\n');

cout<<"Enter the admin-username : ";

string username;

getline(cin,username,'\n');

cout<<"Enter password : ";

string password;

getline(cin,password,'\n');

int i;

for(i = 0;i <noOfAdmins; i++)

{

if(admins[i][0] == id && admins[i][1] == username && admins[i][2] == password)

break;

}

if(i == noOfAdmins)

{

cout<<"Login failed"<<endl;

while(1)

{

cout<<"Press Y to retry, N to return"<<endl;

string temp;

getline(cin,temp,'\n');

if(insensitiveCompare(temp,"y"))

goto ALOG;

else if(insensitiveCompare(temp,"n"))

return;

else

cout<<"invalid input... try again"<<endl;

}

}

else

{

cout<<"Login successful"<<endl;

cout<<"Press any key to continue..."<<endl;

ignore();

admin\_interface();

}

// if success,grant access to admin interface

// else ask to re-enter or return back to admin menu or main menu.

}

void user\_login()

{

ULOG:

// User u1;

CSVProcessor p("user");

string\*\* users = p.getRecords();

/\* for(int i = 0;i < p.countRecords(); i++)

{

for(int j = 0;j < p.countAttributes(); j++)

{

cout<<users[i][j]<<endl;

}

}\*/

int noOfUsers = p.countRecords();

D:

cout<<"Enter the user-id : ";

string user\_id;

getline(cin,user\_id,'\n');

cout<<"Enter the user-username : ";

string username;

getline(cin,username,'\n');

cout<<"Enter password : ";

string password;

getline(cin,password,'\n');

int i;

for(i = 0;i <noOfUsers; i++)

{

if(users[i][0] == user\_id && users[i][1] == username && users[i][2] == password)

{

u1.initialise(users[i]);

break;

}

}

if(i == noOfUsers)

{

cout<<"Login failed"<<endl;

while(1)

{

cout<<"Press Y to retry, N to return"<<endl;

string temp;

getline(cin,temp,'\n');

if(insensitiveCompare(temp,"y"))

goto ULOG;

else if(insensitiveCompare(temp,"n"))

return;

else

cout<<"invalid input... try again"<<endl;

}

}

else

{

cout<<"Login successful"<<endl;

cout<<"Press enter to enter the user's interface..."<<endl;

ignore();

user\_interface();

}

}

void user\_register()

{

User \*u = new User;

u->getData();

u->addUser();

u->createFile();

// delete u;

cout<<"User registered...."<<endl;

cout<<"Press any key to continue"<<endl;

ignore();

}

void admin()

{

while(1)

{

B:

system("cls");

admin\_menu();

int choice;

while(!(cin>>choice))

{

cout<<"Enter integers only"<<endl;

cin.clear();

cin.ignore(10000,'\n');

}

cin.ignore(10000,'\n');

switch(choice)

{

case 1:

admin\_login();

break;

case 2:

return;

case 3:

exit(0);

default:

cout<<"invalid choice!!!"<<endl;

cout<<"Press enter to continue..."<<endl;

ignore();

goto B;

}

}

}

void user()

{

while(1)

{

C:

system("cls");

user\_menu();

int choice;

while(!(cin>>choice))

{

cout<<"Enter integers only"<<endl;

cin.clear();

cin.ignore(10000,'\n');

}

cin.ignore(10000,'\n');

switch(choice)

{

case 1:

user\_login();

break;

case 2:

user\_register();

break;

case 3:

forgot\_password();

case 4:

return;

case 5:

exit(0);

default:

cout<<"invalid choice!!!"<<endl;

cout<<"Press enter to continue..."<<endl;

ignore();

goto C;

}

}

}

void guest()

{

while(1)

{

G:

system("cls");

guest\_menu();

int choice;

while(!(cin>>choice))

{

cout<<"Enter integers only"<<endl;

cin.clear();

cin.ignore(10000,'\n');

}

cin.ignore(10000,'\n');

switch(choice)

{

case 1:

search\_properties();

break;

case 2:

return;

case 3:

exit(0);

default:

cout<<"invalid choice!!!"<<endl;

cout<<"Press enter to continue..."<<endl;

ignore();

goto G;

}

}

}

void search\_properties()

{

string parameters[4];

cout<<"Enter the Location : ";

getline(cin,parameters[0]);

cout<<"Enter the Locality : ";

getline(cin,parameters[1]);

cout<<"Enter the Type of Property you are searching for (flat,plot,house) : ";

getline(cin,parameters[2]);

cout<<"Enter your budget(in lacs) : ";

getline(cin,parameters[3]);

getProperties(parameters);

cout<<"Press enter to continue..."<<endl;

ignore();

// check the records that match the search parameters, and save them in a new temporary file

// display the contents of the temporary file.

}

void add\_property()

{

Property \*p = new Property;

p->getData();

p->addProperty();

cout<<"Property added successfully..."<<endl;

cout<<"Press enter to continue...";

ignore();

}

void delete\_property()

{

DEL:

string id;

cout<<"Enter the property\_id of the property to be deleted : ";

getline(cin,id,'\n');

Property \*p = new Property(id);

if(p->isValid)

{

p->deleteProperty();

cout<<"Property deleted successfully..."<<endl;

cout<<"Press enter to continue...";

ignore();

}

else

{

cout<<"invalid property\_id"<<endl;

while(1)

{

cout<<"Press Y to retry, N to return"<<endl;

string temp;

getline(cin,temp,'\n');

if(insensitiveCompare(temp,"y"))

goto DEL;

else if(insensitiveCompare(temp,"n"))

return;

else

cout<<"invalid input... try again"<<endl;

}

}

}

void update\_property()

{

UPDP:

string id;

cout<<"Enter the property\_id of the property to be updated : ";

getline(cin,id,'\n');

Property \*p = new Property(id);

if(p->isValid)

{

p->updateProperty();

cout<<"Property updated successfully..."<<endl;

cout<<"Press enter to continue...";

ignore();

}

else

{

cout<<"invalid property\_id"<<endl;

while(1)

{

cout<<"Press Y to retry, N to return"<<endl;

string temp;

getline(cin,temp,'\n');

if(insensitiveCompare(temp,"y"))

goto UPDP;

else if(insensitiveCompare(temp,"n"))

return;

else

cout<<"invalid input... try again"<<endl;

}

}

}

void update\_profile()

{

UPDU:

u1.updateProfile();

cout<<"Property updated successfully..."<<endl;

cout<<"Press enter to continue...";

ignore();

}

void purchased\_properties()

{

cout<<"inside purchased properties"<<endl;

u1.myProperties();

/\* system("cls");

CSVProcessor p((\*u1)["username"],1);

cout<<setw(25)<<" ";

int noOfProperties = p.countRecords();

if(noOfProperties == 0)

{

cout<<endl<<endl<<endl;

cout<<setw(25)<<"No properties bought by the user"<<endl;

}

else

{

string \*\*records = p.getRecords();

cout<<".."<<endl;

Property::showMinimalAttributes();

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

{

cout<<setw(25)<<" ";

cout<<setw(25)<<records[i][0];

for(int j = 2;j <= 7; j++)

cout<<setw(25)<<records[i][j];

cout<<endl;

}

}\*/

cout<<"inside my purchased properties"<<endl;

cout<<"Press any key to continue..."<<endl;

ignore();

}

void register\_property()

{

PUR:

Property \*prop = NULL;

cout<<"Enter the property id you want to register : ";

string pid;

getline(cin,pid,'\n');

cout<<"input stored"<<endl;

CSVProcessor p1("property");

cout<<"property opened"<<endl;

string \*\*ppts = p1.getRecords();

cout<<"records stored"<<endl;

bool hit = false;

cout<<"number of records = "<<p1.countRecords()<<endl;

for(int i = 0; i < p1.countRecords(); i++)

{

if(ppts[i][0] == pid)

{

cout<<"Hit occured at "<<i+1<<"th loop"<<endl;

hit = true;

break;

}

}

if(hit == false)

{

cout<<"no such id exists"<<endl;

while(1)

{

cout<<"Press Y to retry, N to return"<<endl;

string temp;

getline(cin,temp,'\n');

if(insensitiveCompare(temp,"y"))

goto PUR;

else if(insensitiveCompare(temp,"n"))

return;

else

cout<<"invalid input... try again"<<endl;

}

}

else

{

prop->detailDisplay();

while(1)

{

cout<<"Press Y to register, N to return"<<endl;

string temp;

getline(cin,temp,'\n');

if(insensitiveCompare(temp,"y"))

{

CSVProcessor p2("registration");

int count = p2.countRecords();

// string row = unparseRecord(matchRecord,p1.countAttributes());

// cout<<row<<endl;

// p2.addRecord(unparseRecord(matchRecord,p1.countAttributes()));

string rid,uid;

if(count == 0)

rid = "1150000001";

else

{

string \*\*records = p2.getRecords();

rid = to\_string(stoi(records[count - 1][0]) + 1);

}

uid = u1["id"];

string row = rid + "," + uid + "," + pid + "," + "P" + "," + "\n";

p2.addRecord(row);

cout<<"request submitted"<<endl;

cout<<"Press any key to continue..."<<endl;

ignore();

return ;

}

else if(insensitiveCompare(temp,"n"))

return;

else

cout<<"invalid input... try again"<<endl;

}

// delete prop;

}

}

void pending\_registrations()

{

system("cls");

CSVProcessor p1("registration");

CSVProcessor p2("user");

CSVProcessor p3("property");

string \*\*registrations = p1.getRecords();

int noOfRegistrations = p1.countRecords();

cout<<endl<<endl<<endl;

cout<<"\t\t\t\t\t\t";

cout<<setw(15)<<"RegistrationID";

cout<<setw(15)<<"UserID";

cout<<setw(15)<<"PropertyID";

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

{

if(registrations[i][3] == "P")

{

cout<<"\t\t\t\t\t\t";

for(int j = 0;j < 3; j++)

cout<<setw(15)<<registrations[i][j];

cout<<endl;

}

}

cout<<endl<<endl;

cout<<"\t\t\t\t\t\t1.Select a property"<<endl;

cout<<"\t\t\t\t\t\t2.Return"<<endl;

int choice;

while(!(cin>>choice))

{

cout<<"Enter integers only"<<endl;

cin.clear();

cin.ignore(10000,'\n');

}

cin.ignore(10000,'\n');

}

void forgot\_password()

{

CSVProcessor p("user");

string \*\*users = p.getRecords();

int noOfUsers = p.countRecords();

FGT:

string user\_id,username,new\_password;

cout<<"Enter the user\_id : ";

getline(cin,user\_id,'\n');

bool hit1,hit2;

int matchIndex;

hit1 = false;

hit2 = false;

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

{

if(users[i][0] == user\_id)

{

matchIndex = i;

hit1 = true;

break;

}

}

cout<<"Enter the username : ";

getline(cin,username,'\n');

if(hit1)

{

if(users[matchIndex][1] == username)

{

hit2 = true;

}

}

// cout<<"hit1 = "<<hit1<<" and hit2 = "<<hit2<<endl;

ignore();

if(hit1 && hit2)

{

cout<<"enter new password : ";

getline(cin,new\_password,'\n');

string \*modified\_user = users[matchIndex];

modified\_user[2].clear();

modified\_user[2] = new\_password;

p.updateRecord(modified\_user[0],unparseRecord(modified\_user,6));

cout<<"Password updated successfully...."<<endl;

cout<<"Press enter to continue....";

ignore();

}

else

{

cout<<"invalid combination of userid and username"<<endl;

cout<<"no such user exists"<<endl;

while(1)

{

cout<<"Press Y to retry, N to return"<<endl;

string temp;

getline(cin,temp,'\n');

if(insensitiveCompare(temp,"y"))

goto FGT;

else if(insensitiveCompare(temp,"n"))

return;

else

cout<<"invalid input... try again"<<endl;

}

}

}

int main()

{

while(1)

{

A:

system("cls");

main\_menu();

int choice;

while(!(cin>>choice))

{

cout<<"Enter integers only"<<endl;

cin.clear();

cin.ignore(10000,'\n');

}

cin.ignore(10000,'\n');

switch(choice)

{

case 1:

admin();

break;

case 2:

user();

break;

case 3:

guest();

break;

case 4:

exit(0);

default:

cout<<"invalid choice!!!"<<endl;

cout<<"Press enter to continue..."<<endl;

ignore();

goto A;

}

}

return 0;

}