#include "stdafx.h"

#include<iostream>

#include<conio.h>

#include<fstream>

#include<conio.h>

using namespace std;

class User;

class Page;

class Post;

class Comment;

class helper

{

public:

static void stringcopy(char\*arr1, const char\* arr2)

{

int i = 0;

while (arr2[i] != '\0')

{

arr1[i] = arr2[i];

i++;

}

arr1[i] = '\0';

}

static void stringCopyWithIndex(char\*arr1, const char\*arr2, int index)

{

int i = 0;

while (arr2[i] != '\0')

{

arr1[index] = arr2[i];

i++;

index++;

}

arr1[index] = '\0';

}

static bool stringComparision(const char\*arr1, const char\*arr2)

{

int index = 0, count = 0;

for (int i = 0; arr1[i] != '\0'; i++)

{

if (arr1[i] == arr2[i])

count++;

}

if (count == strlen(arr1) && count == strlen(arr2))

index = 1;

return index;

}

};

class Date

{

int day;

int month;

int year;

public:

Date() :day(0), month(0), year(0)

{

}

Date(int d, int m, int y) :day(d), month(m), year(y)

{

}

void setDate(int d, int m, int y)

{

day = d;

month = m;

year = y;

}

void printDate()

{

cout << day << "/" << month << "/" << year << endl;

}

int getDay()

{

return day;

}

int getMonth()

{

return month;

}

int getYear()

{

return year;

}

};

class Object

{

char\* Id;

public:

Object() :Id(nullptr)

{

}

Object(char\* \_Id)

{

if (\_Id != nullptr)

{

int size = strlen(\_Id);

Id = new char[size + 1];

helper::stringcopy(Id, \_Id);

}

}

virtual char\* getUserName()

{

return 0;

}

~Object()

{

delete[]Id;

}

void setId(char\*\_Id)

{

if (Id != nullptr)

{

delete[]Id;

}

Id = new char[strlen(\_Id) + 1];

helper::stringcopy(Id, \_Id);

}

char\* getId()

{

return Id;

}

};

class Comment :public Object

{

char \* detail;

Object \* commenter;

public:

Comment() :Object()

{

detail = nullptr;

commenter = nullptr;

}

void setCommenter(Object \* \_commenter)

{

commenter = \_commenter;

}

void setDetail(char\*\_detail)

{

if (detail != nullptr)

delete[]detail;

detail = new char[strlen(\_detail) + 1];

helper::stringcopy(detail, \_detail);

}

char\* getUserName()

{

return commenter->getUserName();

}

char\*getDetail()

{

return detail;

}

~Comment()

{

if (detail != nullptr)

delete[]detail;

}

};

class Post :public Object

{

char\*description;

int noOflikes;

Object\*\*likers;

Date date;

Comment \*\*comments;

int noOfComments;

public:

Post() :description(nullptr), Object(), date(0, 0, 0)

{

likers = nullptr;

comments = nullptr;

}

~Post()

{

delete[]description;

delete[]likers;

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

delete comments[i];

delete[]comments;

}

void setTotalLikes(int j)

{

noOflikes = j;

}

int getNoOfLikes()

{

return noOflikes;

}

void setPostDate(int d, int m, int y)

{

date.setDate(d, m, y);

}

void setPostDescription(char\*\_description)

{

if (description != nullptr)

delete[]description;

description = new char[strlen(\_description) + 1];

helper::stringcopy(description, \_description);

}

char\* getPostDescription()

{

return description;

}

void allocateTotalLikers()

{

likers = new Object\*[10];

}

void setUserPostLikerWithIndex(int index, Object \*liker)

{

likers[index] = liker;

}

void setPagePostLikerWithIndex(int index, Object \*liker)

{

likers[index] = liker;

}

void setComment(char\*\_Id, char\*\_detail, int index, Object\* commenter)

{

comments[index]->setId(\_Id);

comments[index]->setDetail(\_detail);

comments[index]->setCommenter(commenter);

}

void allocateTotalComments()

{

comments = new Comment\*[10];

}

void setNoOfComments(int \_noOfComments)

{

noOfComments = \_noOfComments;

}

int getNoOfComments()

{

return noOfComments;

}

char\* getUsernameWhoCommented(int index)

{

return comments[index]->getUserName();

}

char\* getCommentDetail(int index)

{

return comments[index]->getDetail();

}

void allocateCommentWithIndex(int index)

{

comments[index] = new Comment;

}

char\* getLikerNameWithIndex(int index)

{

return likers[index]->getUserName();

}

int getDay()

{

return date.getDay();

}

int getMonth()

{

return date.getMonth();

}

int getYear()

{

return date.getYear();

}

};

class Page :public Object

{

char \*title;

Post \*\* posts;

int noOfPosts;

int likesCount;

User \* userLiked;

public:

Page() :title(nullptr), Object()

{

likesCount = 0;

posts = nullptr;

userLiked = nullptr;

}

~Page()

{

delete[]title;

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

delete posts[i];

delete[]posts;

}

void setPageId(char\*\_Id)

{

setId(\_Id);

}

void setPageTitle(char\*\_title)

{

if (title != nullptr)

{

delete[]title;

}

title = new char[strlen(\_title) + 1];

helper::stringcopy(title, \_title);

}

void setPostWithIndex(int index, Post\* likedPost)

{

//Deep Copying the posts (Important one)

posts[index]->setId(likedPost->getId());

posts[index]->setPostDate(likedPost->getDay(), likedPost->getMonth(), likedPost->getYear());

posts[index]->setPostDescription(likedPost->getPostDescription());

//posts[j] = likedPost;

}

void setNoOfPosts(int \_noOfPosts)

{

noOfPosts = \_noOfPosts;

}

void allocateTotalPosts()

{

posts = new Post\*[10];//a page can like maxumum 10 posts

//deep copying the posts

for (int i = 0; i < 10; i++)

posts[i] = new Post;

}

char\* getPageTitle()

{

return title;

}

int getNoOfPosts()

{

return noOfPosts;

}

char\* getPostIdWithIndex(int index)

{

return posts[index]->getId();

}

char\* getPostDescriptionWithIndex(int index)

{

return posts[index]->getPostDescription();

}

char\*getUserName() //for polymorphism

{

return title;

}

int getNoOfCommentsWithPostIndex(int index)

{

return posts[index]->getNoOfComments();

}

char\* getUserNameWithPostIndex(int j, int k)

{

return posts[j]->getUsernameWhoCommented(k);

}

char\* getCommentDetailWithPostIndex(int j, int k)

{

return posts[j]->getCommentDetail(k);

}

};

class User :public Object

{

char\*\* friendsIds; //this is for temporary storage.

char\*\* likedPagesIds;//temporary storage

char \*name;

User \*\*friends;

Page \*\* pages;

Post \*\* posts; // The posts on User's Timeline

int noOfFriends;

int noOfpages;

int noOfPosts;

public:

User() :name(nullptr), Object()

{

friends = nullptr;

pages = nullptr;

posts = nullptr;

}

~User()

{

//Note: - friends, pages and posts are just the pointers pointing to some objects. Dont have to delete them

delete[]name;

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

delete[]friendsIds[i];

delete[]friendsIds;

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

delete[]likedPagesIds[i];

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

delete posts[i];

delete[]likedPagesIds;

delete[]friends;

delete[]pages;

delete[]posts;

}

char\* getFriendIdWithIndex(int j)

{

return friendsIds[j];

}

char\*getUserName()

{

return name;

}

char\*getPageIdWithIndex(int j)

{

return likedPagesIds[j];

}

void setUserName(char\*\_name)

{

if (name != nullptr)

delete[]name;

name = new char[strlen(\_name) + 1];

helper::stringcopy(name, \_name);

}

void allocateTtotalFriends()

{

if (friends == nullptr)

friends = new User\*[noOfFriends];

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

//friends[i] = new User;

}

void allocateTotalPages()

{

if (pages == nullptr)

pages = new Page\*[noOfpages];

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

//pages[i] = new Page;

}

void allocateTotalPosts()

{

if (posts == nullptr)

posts = new Post\*[10]; //user can maxumum share 10 posts

//posts should be deep copied

for (int i = 0; i <10; i++)

posts[i] = new Post;

}

int getTotalFriends()

{

return noOfFriends;

}

char\* getFriendNameWithIndex(int index)

{

return friends[index]->getUserName();

}

int getTotalLikedPages()

{

return noOfpages;

}

char\* getPageNameWithIndex(int index)

{

return pages[index]->getPageTitle();

}

void setNoOfPosts(int \_noOfPosts)

{

noOfPosts = \_noOfPosts;

}

void setNoOfFriends(int totalFriends)

{

noOfFriends = totalFriends;

}

void setNoOfPages(int totalPages)

{

noOfpages = totalPages;

}

void allocateTotalFriendIds()

{

friendsIds = new char\*[10]; //user can have maximum 10 friends

}

void allocateTotalLikedPageIds()

{

likedPagesIds = new char\*[10]; //user can have maximum 10 liked pages

}

void setFriendsIdWithIndex(int j, char\*\_Id)

{

friendsIds[j] = new char[strlen(\_Id) + 1];

helper::stringcopy(friendsIds[j], \_Id);

}

void setPageIdWithIndex(int j, char\*\_Id)

{

likedPagesIds[j] = new char[strlen(\_Id) + 1];

helper::stringcopy(likedPagesIds[j], \_Id);

}

void setFriendWithIndex(int j, User\*\_friends)

{

friends[j] = \_friends;

}

void setPageWithIndex(int j, Page\*\_pages)

{

pages[j] = \_pages;

}

void setPostWithIndex(int j, Post \* likedPost)

{

//Deep Copying the posts (Important one)

posts[j]->setId(likedPost->getId());

posts[j]->setPostDate(likedPost->getDay(), likedPost->getMonth(), likedPost->getYear());

posts[j]->setPostDescription(likedPost->getPostDescription());

//posts[j] = likedPost;

}

void viewFriendsList()

{

cout << " - Friend List" << endl;

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

{

cout << friends[i]->getId() << " - " << friends[i]->name << endl;

}

}

void viewLikedPages()

{

cout << " - Liked Pages" << endl;

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

{

cout << pages[i]->getId() << " - " << pages[i]->getPageTitle() << endl;

}

}

char\*getPostIdWithIndex(int index)

{

return posts[index]->getId();

}

int getNoOfPosts()

{

return noOfPosts;

}

char\* getPostDescriptionWithIndex(int index)

{

return posts[index]->getPostDescription();

}

void viewHome()

{

cout << " - Home Page" << endl << endl << endl;

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

{

for (int j = 0; j < friends[i]->getNoOfPosts(); j++)

{

cout << "---" << friends[i]->getUserName() << " shared " << "\"" << friends[i]->getPostDescriptionWithIndex(j) <<"\""<< endl;

for (int k = 0; k < friends[i]->posts[j]->getNoOfComments(); k++)

{

cout << friends[i]->posts[j]->getUsernameWhoCommented(k) << "\t" << " commented " << friends[i]->posts[j]->getCommentDetail(k) << endl;

}

cout << endl << endl;

}

}

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

{

for (int j = 0; j < pages[i]->getNoOfPosts(); j++)

{

cout << "---" << pages[i]->getPageTitle() << " shared " << "\"" << pages[i]->getPostDescriptionWithIndex(j) << "\"" << endl;

for (int k = 0; k < pages[i]->getNoOfCommentsWithPostIndex(j); k++)

{

cout << pages[i]->getUserNameWithPostIndex(j, k) << "\t" << " commented " << pages[i]->getCommentDetailWithPostIndex(j, k) << endl;

}

cout << endl << endl;

}

}

}

};

class Facebook

{

User\*\* users;

Page\*\* pages;

Post\*\* posts;

User\* currentUser;

Date CurrentDate;

static int totalUsers;

static int totalPages;

static int totalPosts;

static int totalComments;

public:

void setCurrentDate(int d, int m, int y)

{

CurrentDate.setDate(d, m, y);

}

Facebook()

{

//pointing everything to nullptr

currentUser = nullptr;

pages = nullptr;

}

void loadData()

{

ifstream fin;

int j = 0, pagesCount = 0;

char buffer[130];

char buffer2[80];

char buffer3[80];

int \*index, \*index2;

int total;

fin.open("Pages.txt"); //reading pages

if (fin.is\_open())

{

fin >> total; //total no of pages

totalPages = total;

//fin.getline(buffer, 80); //get into next line

pages = new Page\*[total];

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

{

pages[i] = new Page;

fin >> buffer; //id of page;

pages[i]->setId(buffer);

fin.getline(buffer, 80);//name of page

//removing '\t';

for (int i = 0; buffer[i] != '\0'; i++)

{

buffer[i] = buffer[i + 1];

}

pages[i]->setPageTitle(buffer);

}

}

else

cout << "Error reading pages file.\n";

fin.close();

fin.open("Users.txt"); //reading users

if (fin.is\_open())

{

fin >> total;

totalUsers = total;

//fin.getline(buffer, 80); //coming into next line

users = new User\*[total];

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

{

users[i] = new User;

fin >> buffer;

users[i]->setId(buffer);

fin >> buffer2; //first name

helper::stringcopy(buffer, buffer2); //copying first name

int size = strlen(buffer);

buffer[size] = ' ';

buffer[size + 1] = '\0';

fin >> buffer2; //second second name

helper::stringCopyWithIndex(buffer, buffer2, strlen(buffer)); //fullname using string concatenate

users[i]->setUserName(buffer);

//allocating space for saving temporary ids

users[i]->allocateTotalFriendIds();

users[i]->allocateTotalLikedPageIds();

fin >> buffer;

j = 0;

while (buffer[0] != '-' || buffer[1] != '1')

{

users[i]->setFriendsIdWithIndex(j, buffer);

j++;

fin >> buffer;

}

users[i]->setNoOfFriends(j);//setting number of friends

j = 0;

fin >> buffer;

while (buffer[0] != '-' || buffer[1] != '1')

{

users[i]->setPageIdWithIndex(j, buffer);

j++;

fin >> buffer;

pagesCount++;

}

users[i]->setNoOfPages(j);

fin.getline(buffer, 80); //moving to next line

}

}

else

cout << "Error opening Users file";

fin.close();

fin.open("Posts.txt"); //reading posts

if (fin.is\_open())

{

index = new int[totalUsers];

index2 = new int[totalPages];

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

{

index[i] = 0;

users[i]->allocateTotalPosts();//allocating total no of posts shared by user

}

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

{

index2[i] = 0;

pages[i]->allocateTotalPosts();//page can maximum like 10 post

}

fin >> total;

posts = new Post\*[total]; //total number of posts

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

posts[i] = new Post;

totalPosts = total;

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

{

fin >> buffer; //post Id

posts[i]->setId(buffer); //setting post id

fin.getline(buffer, 80); //move to line[01]

int d, m, y;

fin >> d >> m >> y;

posts[i]->setPostDate(d, m, y);

fin.getline(buffer, 80); //moving into line[02]

fin.getline(buffer, 80); //reading description

posts[i]->setPostDescription(buffer);

fin.getline(buffer, 80); //reading id which shared posts.

if (buffer[0] == 'u')

{

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

{

if (helper::stringComparision(buffer, users[j]->getId()) == 1)

{

users[j]->setPostWithIndex(index[j], posts[i]);

index[j]++;

users[j]->setNoOfPosts(index[j]);

break;

}

}

}

else if (buffer[0] == 'p')

{

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

{

if (helper::stringComparision(buffer, pages[j]->getId()) == 1)

{

pages[j]->setPostWithIndex(index2[j], posts[i]);

index2[j]++;

pages[j]->setNoOfPosts(index2[j]);

break;

}

}

}

posts[i]->allocateTotalLikers(); //allocating likers to 10

fin >> buffer;

j = 0;

while (buffer[0] != '-' || buffer[1] != '1')

{

if (buffer[0] == 'u')

{

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

{

if (helper::stringComparision(buffer, users[k]->getId()) == 1)

{

posts[i]->setUserPostLikerWithIndex(j, users[k]);

j++;

break;

}

}

}

else if (buffer[0] == 'p')

{

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

{

if (helper::stringComparision(buffer, pages[k]->getId()) == 1)

{

posts[i]->setPagePostLikerWithIndex(j, pages[k]);

j++;

break;

}

}

}

fin >> buffer;

}

posts[i]->setTotalLikes(j);

fin.getline(buffer, 80); //moving into starting line of next post

}

delete[]index;

delete[]index2;

} // end of opened file

else

cout << "Error Opening Posts File.\n";

fin.close();

fin.open("Comments.txt"); //reading comments

if (fin.is\_open())

{

fin >> totalComments;

index = new int[totalPosts]; //to save index of comments on each posts

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

index[i] = 0;

for (int p = 0; p<totalPosts; p++)

posts[p]->allocateTotalComments(); //total comments can be 10

while (!fin.eof())

{

Comment \*comment = new Comment;

fin >> buffer;

comment->setId(buffer);

fin >> buffer;

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

{

if (helper::stringComparision(buffer, posts[i]->getId()) == 1)

{

fin >> buffer;

if (buffer[0] == 'u')

{

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

{

if(helper::stringComparision(buffer,users[j]->getId())==1)

{

fin.getline(buffer, 80);

posts[i]->allocateCommentWithIndex(index[i]);

posts[i]->setComment(comment->getId(),buffer,index[i], users[j]); //compositing comment detail, user who did that, comment id and post on which comment was happened

index[i]++;

posts[i]->setNoOfComments(index[i]);

break;

}

}

}

else if (buffer[0] == 'p')

{

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

{

if (helper::stringComparision(buffer, pages[j]->getId()) == 1)

{

fin.getline(buffer, 80);

posts[i]->allocateCommentWithIndex(index[i]);

posts[i]->setComment(comment->getId(), buffer, index[i], pages[j]);

index[i]++;

posts[i]->setNoOfComments(index[i]);

break;

}

}

}

}

} // end of for loop

delete comment; //deallocating temporary one

} // end of file

}

else

cout << "Error Opening Comments File.\n";

}

void linkData()

{

char \*buffer;

char\* temp;

int totalFriends;

int totalLikedPages;

for (int i = 0; i < totalUsers; i++) //for setting every user friends

{

users[i]->allocateTtotalFriends();

users[i]->allocateTotalPages();

totalFriends = users[i]->getTotalFriends();

totalLikedPages = users[i]->getTotalLikedPages();

for (int j = 0; j < totalFriends; j++) //checking for each friend

{

buffer = users[i]->getFriendIdWithIndex(j);

for (int k = 0; k < totalUsers; k++) //checking each friend in k users

{

temp = users[k]->getId();

if (helper::stringComparision(buffer, temp) == 1)

{

users[i]->setFriendWithIndex(j, users[k]);

}

}

}

for (int j = 0; j < totalLikedPages; j++) //checking for each page

{

buffer = users[i]->getPageIdWithIndex(j);

for (int k = 0; k < totalPages; k++) //checking each page in k Pages

{

temp = pages[k]->getId();

if (helper::stringComparision(buffer, temp) == 1)

{

users[i]->setPageWithIndex(j, pages[k]);

}

}

}

}

}

bool setCurrentUser()

{

int flag = 0;

char \_Id[5] = "u7";

for (int i = 0; i < totalUsers; i++)//traversing to select the user

{

if (helper::stringComparision(users[i]->getId(), \_Id) == 1)

{

currentUser = users[i];

flag = 1;

}

}

return flag;

}

void run()

{

char buffer[80], buffer2[80];

//cout << "Today's Date:\t";

//CurrentDate.printDate();

if (setCurrentUser() == 1)

{

cout << currentUser->getUserName() << " successfully set as Current User" << endl << endl;

CurrentDate.setDate(15, 11, 2017);

cout << "System Date: ";

CurrentDate.printDate();

cout << currentUser->getUserName();

currentUser->viewFriendsList();

cout << currentUser->getUserName();

currentUser->viewLikedPages();

cout << currentUser->getUserName();

currentUser->viewHome();

/\*

cout << "Enter the Id Of the post you want to check likes:\t";

cin >> buffer;

checkLikersOnSpecificPost(buffer);

cout << "Enter the Post Id on which you want to comment:\t";

cin >> buffer;

cout << "Enter the Comment:\t";

cin >> buffer2;

addCommentOnThePost(buffer, buffer2);

currentUser->viewHome();

\*/

}

else

cout << "\nUser Not found!\n";

}

void checkLikersOnSpecificPost(char\* buffer)

{

int flag = 0;

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

{

if (helper::stringComparision(buffer, posts[i]->getId()) == 1) //searching post

{

if (posts[i]->getNoOfLikes() > 0) //likes check

{

cout << "Likers of " << buffer << " are:\n";

for (int j = 0; j < posts[i]->getNoOfLikes(); j++)

{

flag = 1;

cout << posts[i]->getLikerNameWithIndex(j) << endl; //liker name

}

break;

}

}

}

if (flag == 0)

cout << "No Likes on the Post.\n";

}

void addCommentOnThePost(char \* buffer, char\* buffer2)

{

char buffer3[6];

//setting comment id in buffer3

buffer3[0] = 'c';

int mod, comments = totalComments + 1;

if (totalComments > 9)

{

mod = (comments % 10) + 304;

comments = (comments / 10) + 304;

buffer3[2] = (char)mod;

buffer3[1] = (char)comments;

buffer3[3] = '\0';

}

else

{

buffer3[1] = comments;

buffer3[2] = '\0';

}

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

{

if (helper::stringComparision(buffer, posts[i]->getId()) == 1)

{

int n = posts[i]->getNoOfComments();

if (n < 10) //comments check;

{

posts[i]->allocateCommentWithIndex(n + 1);

posts[i]->setComment(buffer3, buffer2, n + 1, currentUser);

posts[i]->setNoOfComments(n + 1);

break;

}

else

cout << "Already 10 comments on the post";

}

}

}

~Facebook()

{

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

delete users[i];

delete[]users;

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

delete pages[i];

delete[]pages;

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

delete posts[i];

delete[]posts;

}

};

int Facebook::totalPages = 0;

int Facebook::totalPosts = 0;

int Facebook::totalUsers = 0;

int Facebook::totalComments = 0;

int main()

{

Facebook fb;

fb.setCurrentDate(15, 11, 2017);

fb.loadData();//loading pages, users and posts

fb.linkData();

fb.run();

\_getch();

return 0;

}