using namespace std;

class User;

class Page;

class Post;

class Comment;

class helper

{

public:

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

{

int flag = 1;

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

{

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

{

flag = 0;

break;

}

}

return flag;

}

static void myStringCopy(char \* temp1, const char \* temp2)

{

int i = 0;

while (temp2[i] != '\0')

{

temp1[i] = temp2[i];

i++;

}

temp1[i] = '\0';

}

};

class Date

{

int day, month, year;

public:

Date(int d = 0, int m = 0, int y = 0) // default and parameterized cons

{

day = d;

month = m;

year = y;

}

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

{

day = d;

month = m;

year = y;

}

int getDay()

{

return day;

}

int getYear()

{

return year;

}

void getYear(int y)

{

year = y;

}

void setDay(int d)

{

day = d;

}

int getMonth()

{

return month;

}

void setMonth(int m)

{

month = m;

}

void printDate()

{

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

}

};

class Object

{

char \* Identity;

public:

Object(char\* id)

{

if (id != nullptr)

{

int size = strlen(id);

Identity = new char[size + 1];

helper::myStringCopy(Identity, id);

}

}

Object()

{

Identity = nullptr;

}

~Object()

{

delete[]Identity;

}

virtual char \* getUserName()

{

return 0;

}

char \* getId()

{

return Identity;

}

void setId(char \* id)

{

if (Identity != nullptr)

{

delete[] Identity;

}

Identity = new char[strlen(id) + 1];

helper::myStringCopy(Identity, id);

}

};

class Comment :public Object

{

char \* comment;

Object \* Commenter;

public:

Comment() : Object()

{

comment = nullptr;

Commenter = nullptr;

}

~Comment()

{

if (comment != nullptr)

{

delete[]comment;

}

}

char \* getUserName()

{

return Commenter->getUserName();

}

void setComment(char \* comt)

{

if (comment != nullptr)

{

delete[]comment;

}

comment = new char[strlen(comt) + 1];

helper::myStringCopy(comment, comt);

}

void setCommenter(Object \* comtr)

{

Commenter = comtr;

}

char \* getComment()

{

return comment;

}

};

class Post :public Object

{

Date date;

Comment \*\* Comments;

Object \*\* Likers;

char \* postInfo;

int likesCount, commentsCount;

public:

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

{

postInfo = nullptr;

Likers = nullptr;

Comments = nullptr;

}

~Post()

{

delete[]postInfo;

delete[]Likers;

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

{

delete Comments[i];

}

delete[]Comments;

}

int getNoOfLikes()

{

return likesCount;

}

void setTotalLikes(int j)

{

likesCount = j;

}

void setPostInfo(char\*\_description)

{

if (postInfo != nullptr)

delete[]postInfo;

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

helper::myStringCopy(postInfo, \_description);

}

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

{

date.setDate(d, m, y);

}

char \* getPostInfo()

{

return postInfo;

}

void allocLikers()

{

Likers = new Object \* [10];

}

void allocComments()

{

Comments = new Comment \*[10];

}

void uPostLike(int n, Object \* liked)

{

Likers[n] = liked;

}

void pPostLiker(int n, Object \*liked)

{

Likers[n] = liked;

}

void setComments(int x)

{

commentsCount = x;

}

void setComment(char\* id, char\* info, int n, Object\* cmtr)

{

Comments[n]->setCommenter(cmtr);

Comments[n]->setId(id);

Comments[n]->setComment(info);

}

int getComments()

{

return commentsCount;

}

char\* getUserCmtname(int n)

{

return Comments[n]->getUserName();

}

char\* getLikerNameWithIndex(int n)

{

return Likers[n]->getUserName();

}

void CommentWithIndex(int index)

{

Comments[index] = new Comment;

}

char \* getComInfo(int n)

{

return Comments[n]->getComment();

}

char \* getIdwithIndex(int n)

{

return Likers[n]->getId();

}

int getYear()

{

return date.getYear();

}

int getDay()

{

return date.getDay();

}

int getMonth()

{

return date.getMonth();

}

};

class Page :public Object

{

char \* Name;

Post \*\* Posts;

int PostsCount;

int likesCount;

User \* userLiked;

public:

Page() : Object()

{

Name = nullptr;

Posts = nullptr;

userLiked = nullptr;

likesCount = 0;

}

~Page()

{

delete[]Name;

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

{

delete Posts[i];

}

delete[]Posts;

}

void setPageId(char \* id)

{

setId(id);

}

void setPageName(char\* name)

{

if (Name != nullptr)

{

delete[]Name;

}

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

helper::myStringCopy(Name, name);

}

void setPostWithIndex(int n, Post \* likedPost)

{

Posts[n]->setId(likedPost->getId());

Posts[n]->setPostDate(likedPost->getDay(), likedPost->getMonth(), likedPost->getYear());

Posts[n]->setPostInfo(likedPost->getPostInfo());

}

void setPosts(int x)

{

PostsCount = x;

}

void allocPosts()

{

Posts = new Post \* [10]; for (int i = 0; i < 10; i++)

{

Posts[i] = new Post;

}

}

char\* getPageName()

{

return Name;

}

int getPostCnt()

{

return PostsCount;

}

char\* getPostIdWithN(int n)

{

return Posts[n]->getId();

}

char\* getPostInfoWithN(int n)

{

return Posts[n]->getPostInfo();

}

char \* getUserName()

{

return Name;

}

};

class User :public Object

{

char \* Name;

User \*\* Friend;

char\*\* friendIdentity;

Page \*\* Pages;

Post \*\* Posts;

char \*\* IDlikedPages;

int FriendsCount;

int PostsCount;

int PagesCount;

public:

User() : Object()

{

Name = nullptr;

Friend = nullptr;

Pages = nullptr;

Posts = nullptr;

}

~User()

{

delete[]Name;

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

{

delete[] friendIdentity[i];

}

delete[]friendIdentity;

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

{

delete[] IDlikedPages[i];

}

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

{

delete Posts[i];

}

delete[] IDlikedPages;

delete[] Friend;

delete[] Pages;

delete[] Posts;

}

char \* getPageIdWithIndex(int j)

{

return IDlikedPages[j];

}

char\* getFriendIdWithIndex(int j)

{

return friendIdentity[j];

}

char\*getUserName()

{

return Name;

}

void allocateTotalPosts()

{

if (Posts == nullptr)

{

Posts = new Post\*[10];

}

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

{

Posts[i] = new Post;

}

}

void allocateTotalPages()

{

if (Pages == nullptr)

{

Pages = new Page\*[PagesCount];

}

}

void setUserName(char\*\_name)

{

if (Name != nullptr)

delete[]Name;

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

helper::myStringCopy(Name, \_name);

}

void allocateTtotalFriends()

{

if (Friend == nullptr)

{

Friend = new User\*[FriendsCount];

}

}

int getTotalFriends()

{

return FriendsCount;

}

char\* getPageNameWithIndex(int index)

{

return Pages[index]->getPageName();

}

void setNoOfPosts(int \_noOfPosts)

{

PostsCount = \_noOfPosts;

}

char \* getFriendNameWithIndex(int n)

{

return Friend[n]->getUserName();

}

int getTotalLikedPages()

{

return PagesCount;

}

void setNoOfFriends(int totalFriends)

{

FriendsCount = totalFriends;

}

void allocateTotalFriendIds()

{

friendIdentity = new char\*[10];

}

void allocateTotalLikedPageIds()

{

IDlikedPages = new char\*[10];

}

void setNoOfPages(int tPag)

{

PagesCount = tPag;

}

void setFriendsIdWithIndex(int j, char \* id)

{

friendIdentity[j] = new char[strlen(id) + 1];

helper::myStringCopy(friendIdentity[j], id);

}

void setPageIdWithIndex(int j, char \* id)

{

IDlikedPages[j] = new char[strlen(id) + 1];

helper::myStringCopy(IDlikedPages[j], id);

}

void setPageWithIndex(int j, Page\* pg)

{

Pages[j] = pg;

}

void setFriendWithIndex(int j, User \* fnd )

{

Friend[j] = fnd;

}

void setPostWithIndex(int j, Post \* likedPost)

{

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

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

Posts[j]->setPostInfo(likedPost->getPostInfo());

}

void viewFriendsList()

{

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

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

{

cout << Friend[i]->getId() << " - " << Friend[i]->Name << endl;

}

}

void viewLikedPages()

{

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

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

{

cout << Pages[i]->getId() << " - " << Pages[i]->getPageName() << endl;

}

}

char \* getPostIdWithIndex(int n)

{

return Posts[n]->getId();

}

int getNoOfPosts()

{

return PostsCount;

}

char \* getPostDescriptionWithIndex(int n)

{

return Posts[n]->getPostInfo();

}

void HomePage()

{

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

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

{

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

{

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

cout << endl << endl;

}

}

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

{

for (int j = 0; j < Pages[i]->getPostCnt(); j++)

{

cout << "---" << Pages[i]->getPageName() << " shared " << "\"" << Pages[i]->getPostInfoWithN(j) << "\"" << endl;

cout << endl << endl;

}

}

}

};

class SocialNetwork

{

static int totalPages;

static int totalPosts;

static int totalUsers;

static int totalComments;

User \*\* users;

Page \*\* pages;

Post \*\* posts;

User \* currentUser;

Date CurrentDate;

public:

SocialNetwork()

{

currentUser = nullptr;

pages = nullptr;

}

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

{

CurrentDate.setDate(d, m, y);

}

void readDataFromFile()

{

ifstream fin;

int j = 0, pagesCount = 0;

int total;

int \* N1, \* N2;

char temp1[130];

char temp2[80];

char temp3[80];

fin.open("Pages.txt");

if (fin.is\_open())

{

fin >> total;

totalPages = total;

pages = new Page\*[total];

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

{

pages[i] = new Page;

fin >> temp1;

pages[i]->setId(temp1);

fin.getline(temp1, 80);

for (int i = 0; temp1[i] != '\0'; i++)

{

temp1[i] = temp1[i + 1];

}

pages[i]->setPageName(temp1);

}

}

else

cout << "Error opening file";

fin.close();

fin.open("Users.txt");

if (fin.is\_open())

{

fin >> total;

totalUsers = total;

users = new User\*[total];

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

{

users[i] = new User;

fin >> temp1; //id

users[i]->setId(temp1);

fin >> temp2; // f name

helper::myStringCopy(temp1, temp2);

int size = strlen(temp1);

temp1[size] = ' ';

temp1[size + 1] = '\0';

fin >> temp2; //s name

strcat\_s(temp1, temp2);

users[i]->setUserName(temp1);

users[i]->allocateTotalLikedPageIds();

users[i]->allocateTotalFriendIds();

fin >> temp1;

j = 0;

while (temp1[0] != '-' || temp1[1] != '1')

{

users[i]->setFriendsIdWithIndex(j, temp1);

j++;

fin >> temp1;

}

users[i]->setNoOfFriends(j);

j = 0;

fin >> temp1;

while (temp1[0] != '-' || temp1[1] != '1')

{

users[i]->setPageIdWithIndex(j, temp1);

j++;

fin >> temp1;

pagesCount++;

}

users[i]->setNoOfPages(j);

fin.getline(temp1, 80);//moving to next line

}

}

else

cout << "Error opening Users file";

fin.close();

fin.open("Posts.txt");

if (fin.is\_open())

{

N1 = new int[totalUsers];

N2 = new int[totalPages];

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

{

N1[i] = 0;

users[i]->allocateTotalPosts();

}

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

{

N2[i] = 0;

pages[i]->allocPosts();

}

fin >> total;

posts = new Post\*[total];

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

{

posts[i] = new Post;

}

totalPosts = total;

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

{

fin >> temp1;

posts[i]->setId(temp1);

fin.getline(temp1, 80);

int date, month, year;

fin >> date >> month >> year;

posts[i]->setPostDate(date, month, year);

fin.getline(temp1, 80);

fin.getline(temp1, 80);

posts[i]->setPostInfo(temp1);

fin.getline(temp1, 80);

if (temp1[0] == 'u')

{

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

{

if (helper::stringComparision(temp1, users[j]->getId()) == 1)

{

users[j]->setPostWithIndex(N1[j], posts[i]);

N1[j]++;

users[j]->setNoOfPosts(N1[j]);

break;

}

}

}

else if (temp1[0] == 'p')

{

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

{

if (helper::stringComparision(temp1, pages[j]->getId()) == 1)

{

pages[j]->setPostWithIndex(N2[j], posts[i]);

N2[j]++;

pages[j]->setPosts(N2[j]);

break;

}

}

}

posts[i]->allocLikers();

fin >> temp1;

j = 0;

while (temp1[0] != '-' ||temp1[1] != '1')

{

if (temp1[0] == 'u')

{

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

{

if (helper::stringComparision(temp1, users[k]->getId()) == 1)

{

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

j++;

break;

}

}

}

else if (temp1[0] == 'p')

{

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

{

if (helper::stringComparision(temp1, pages[k]->getId()) == 1)

{

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

j++;

break;

}

}

}

fin >> temp1;

}

posts[i]->setTotalLikes(j);

fin.getline(temp1, 80);

}

delete[]N1;

delete[]N2;

}

Else{ cout << "Error Opening File";} fin .close();

fin.open("Comments.txt");

if (fin.is\_open())

{

fin >> totalComments;

N1 = new int[totalPosts];

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

{

N1[i] = 0;

}

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

{

posts[p]->allocComments();

}

while (!fin.eof())

{

Comment \* comment = new Comment;

fin >> temp1;

comment->setId(temp1);

fin >> temp1;

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

{

if (helper::stringComparision(temp1, posts[i]->getId()) == 1)

{

fin >> temp1;

if (temp1[0] == 'u')

{

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

{

if (helper::stringComparision(temp1, users[j]->getId()) == 1)

{

fin.getline(temp1, 80);

posts[i]->CommentWithIndex(N1[i]);

posts[i]->setComment(comment->getId(), temp1, N1[i], users[j]);

N1[i]++;

posts[i]->setComments(N1[i]);

break;

}

}

}

else if (temp1[0] == 'p')

{

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

{

if (helper::stringComparision(temp1, pages[j]->getId()) == 1)

{

fin.getline(temp1, 80);

posts[i]->CommentWithIndex(N1[i]);

posts[i]->setComment(comment->getId(), temp1, N1[i], pages[j]);

N1[i]++;

posts[i]->setComments(N1[i]);

break;

}

}

}

}

}

delete comment;

}

}

else

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

}

void RelatingData()

{

char \* tempbuff;

char \* temp;

int totalFriends;

int totalLikedPages;

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

{

users[i]->allocateTtotalFriends();

users[i]->allocateTotalPages();

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

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

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

{

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

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

{

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

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

{

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

}

}

}

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

{

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

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

{

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

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

{

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

}

}

}

}

}

bool setCurrentUser()

{

int flag = 0;

char \_Id[5] = "u7";

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

{

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

{

currentUser = users[i];

flag = 1;

}

}

return flag;

}

void run()

{

char buffer[80], buffer2[80];

if (setCurrentUser() == 1)

{

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

CurrentDate.setDate(15, 11, 2017);

cout << "System Date: ";

CurrentDate.printDate();

cout << endl << endl << "-------------------------------------------------------------------" << endl;

cout << currentUser->getUserName();

currentUser->viewFriendsList();

cout << endl << endl << "-------------------------------------------------------------------" << endl;

cout << currentUser->getUserName();

currentUser->viewLikedPages();

cout << endl << endl << "-------------------------------------------------------------------" << endl;

cout << currentUser->getUserName();

currentUser->HomePage();

cout << endl << endl << "-------------------------------------------------------------------" << endl;

// timeline

//cout << " Timeline " << endl << endl;

cout << " View Liked Post " << endl << endl;

//cin >> buffer;

checkLikers("post5");

cout << endl << endl << "-------------------------------------------------------------------" << endl;

cout << " Like Post " << endl << endl;

cout << " View Liked Post " << endl << endl;

checkLikers("post5");

cout << endl << endl << "-------------------------------------------------------------------" << endl;

}

else

cout << endl << "User Not found" << endl;

}

void checkLikers(char\* buffer)

{

cout << "Post Liked By:" << endl << endl;

int flag = 0;

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

{

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

{

if (posts[i]->getNoOfLikes() > 0)

{

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

{

flag = 1;

cout << posts[i]->getIdwithIndex(j) << " -- " << posts[i]->getLikerNameWithIndex(j) << endl;

}

break;

}

}

}

if (flag == 0)

{

cout << "0 Likes" << endl;

}

}

~SocialNetwork()

{

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 SocialNetwork::totalPages = 0;

int SocialNetwork::totalPosts = 0;

int SocialNetwork::totalUsers = 0;

int SocialNetwork::totalComments = 0;

int main() { SocialNetwork obj; obj.readDataFromFile();obj.RelatingData();obj.run();getc}