**Контрольные примеры:**

======================================== | Choose an option | |--------------------------------------| | 0. Exit | | 1. Print all users | | 2. Print all professions | | 3. Add new profession | | 4. Add new user | | 5. Update user data | | 6. Filter users | | 7. Sort users | | 8. Delete profession | | 9. Delete user | | 10. Clear user list | | 11. Clear profession list | ======================================== Option: 1

======================================== | Option: Print all users | ======================================== ================================================================================================================================= | ID | Full Name | Age | Profession | Friends Rating | Public Rating | Friends count | Friends IDs | |----|------------------------|-----|------------------|----------------|---------------|---------------|-----------------------| | 1 | John Doe | 10 | undefined | 4.5 | 3.9 | 1 | 2 | | 2 | Jane Doe | 20 | undefined | 4.0 | 4.0 | 1 | 1 | | 3 | Alice Johnson | 28 | pilot | 4.2 | 3.7 | 4 | 1, 2, 6, 8 | | 4 | Sarah Taylor | 31 | teacher | 4.0 | 4.1 | 5 | 8, 5, 6, 3, 1 | | 5 | Robert White | 29 | dentist | 4.3 | 3.8 | 3 | 1, 2, 3 | | 6 | Michael Brown | 33 | engineer | 3.9 | 4.0 | 5 | 3, 6, 9, 10, 2 | | 7 | Linda Martinez | 32 | pilot | 3.9 | 3.7 | 4 | 4, 6, 5, 1 | | 8 | Jane Smith | 25 | driver | 3.8 | 4.1 | 2 | 1, 3 | | 9 | Jack London | 31 | writer | 5.0 | 5.0 | 6 | 8, 5, 6, 3, 1, 9 | | 10 | Emily Davis | 27 | driver | 4.1 | 3.8 | 3 | 1, 2, 3 | | 11 | David Wilson | 35 | actor | 4.0 | 4.2 | 2 | 5, 2 | | 12 | Yakui The Maid | 35 | musician | 5.0 | 4.0 | 1 | 9 | | 13 | God is an Astronaut | 20 | musician | 5.0 | 5.0 | 2 | 1, 12 | ================================================================================================================================= Press ENTER to continue

======================================== | Option: Print all professions | ======================================== ======================================== | ID | Name | |----|---------------------------------| | 1 | pilot | | 2 | engineer | | 3 | teacher | | 4 | driver | | 5 | dentist | | 6 | actor | | 7 | writer | | 8 | musician | | 9 | test long name of new profes... | ======================================== Press ENTER to continue

======================================== | Option: Add new profession | ======================================== Enter profession name: new\_profession Success: profession added ======================================== | ID | Name | |----|---------------------------------| | 10 | new\_profession | ======================================== Press ENTER to continue

======================================== | Option: Add new user | ======================================== Enter information for new user: Enter user name: new user Success: name specified Enter user age: -10 Failed: invalid or impossible age Enter user friends rating: 2 Success: friends rating specified Enter user public rating: 10 Failed: invalid or impossible rating Enter user friends count (less than 13): 12 ================================================================================================================================= | ID | Full Name | Age | Profession | Friends Rating | Public Rating | Friends count | Friends IDs | |----|------------------------|-----|------------------|----------------|---------------|---------------|-----------------------| | 1 | John Doe | 10 | undefined | 4.5 | 3.9 | 1 | 2 | | 2 | Jane Doe | 20 | undefined | 4.0 | 4.0 | 1 | 1 | | 3 | Alice Johnson | 28 | pilot | 4.2 | 3.7 | 4 | 1, 2, 6, 8 | | 4 | Sarah Taylor | 31 | teacher | 4.0 | 4.1 | 5 | 8, 5, 6, 3, 1 | | 5 | Robert White | 29 | dentist | 4.3 | 3.8 | 3 | 1, 2, 3 | | 6 | Michael Brown | 33 | engineer | 3.9 | 4.0 | 5 | 3, 6, 9, 10, 2 | | 7 | Linda Martinez | 32 | pilot | 3.9 | 3.7 | 4 | 4, 6, 5, 1 | | 8 | Jane Smith | 25 | driver | 3.8 | 4.1 | 2 | 1, 3 | | 9 | Jack London | 31 | writer | 5.0 | 5.0 | 6 | 8, 5, 6, 3, 1, 9 | | 10 | Emily Davis | 27 | driver | 4.1 | 3.8 | 3 | 1, 2, 3 | | 11 | David Wilson | 35 | actor | 4.0 | 4.2 | 2 | 5, 2 | | 12 | Yakui The Maid | 35 | musician | 5.0 | 4.0 | 1 | 9 | | 13 | God is an Astronaut | 20 | musician | 5.0 | 5.0 | 2 | 1, 12 | ================================================================================================================================= Success: friends count specified Enter user friends ids Example: 1,2,3,4,5 Enter friends ids: 1,1,2,3,1,-10,1234,1,5 It seems that the number of entered IDs does not correspond to the specified number of friends updating friends count: 9 Duplicated ID: 1 Duplicated ID: 1 Duplicated ID: 1 It seems that some IDs are entered more than once -> updating friends count: 6 ID not found: -10 ID not found: 1234 It seems that list of users does not contain some of entered IDs -> updating friends count: 4 Success: friends ids specified ======================================== | ID | Name | |----|---------------------------------| | 1 | pilot | | 2 | engineer | | 3 | teacher | | 4 | driver | | 5 | dentist | | 6 | actor | | 7 | writer | | 8 | musician | | 9 | test long name of new profes... | ======================================== Enter profession id: -2 Failed: profession not found Success: user has been added! ================================================================================================================================= | ID | Full Name | Age | Profession | Friends Rating | Public Rating | Friends count | Friends IDs | |----|------------------------|-----|------------------|----------------|---------------|---------------|-----------------------| | 14 | new user | 0 | undefined | 2.0 | 0.0 | 4 | 1, 2, 3, 5 | ================================================================================================================================= Press ENTER to continue

======================================== | Option: Update user data | ======================================== ================================================================================================================================= | ID | Full Name | Age | Profession | Friends Rating | Public Rating | Friends count | Friends IDs | |----|------------------------|-----|------------------|----------------|---------------|---------------|-----------------------| | 3 | Alice Johnson | 28 | pilot | 4.2 | 3.7 | 4 | 1, 2, 6, 8 | ================================================================================================================================= Which field do you want to edit? 1. full name 2. age 3. profession 4. friends rating 5. public rating 6. friends 7. all fields Enter option: 2 ======================================== | Option: Specify user age | ======================================== Enter user age: 30 Success: age specified Updated user: ================================================================================================================================= | ID | Full Name | Age | Profession | Friends Rating | Public Rating | Friends count | Friends IDs | |----|------------------------|-----|------------------|----------------|---------------|---------------|-----------------------| | 3 | Alice Johnson | 30 | pilot | 4.2 | 3.7 | 4 | 1, 2, 6, 8 | ================================================================================================================================= Press ENTER to continue

======================================== | Option: Filter users | ======================================== 1. Name 2. Profession 3. Age 4. Friends Rating 5. Public Rating 6. Friends Count Enter option: 1 Enter name: ja ================================================================================================================================= | ID | Full Name | Age | Profession | Friends Rating | Public Rating | Friends count | Friends IDs | |----|------------------------|-----|------------------|----------------|---------------|---------------|-----------------------| | 2 | Jane Doe | 20 | undefined | 4.0 | 4.0 | 1 | 1 | | 8 | Jane Smith | 25 | driver | 3.8 | 4.1 | 2 | 1, 3 | | 9 | Jack London | 31 | writer | 5.0 | 5.0 | 6 | 8, 5, 6, 3, 1, 9 | ================================================================================================================================= Press ENTER to continue

======================================== | Option: Sort users | ======================================== 1. Sort by id 2. Sort by name 3. Sort by age 4. Sort by friends rating 5. Sort by public rating 6. Sort by friends count Enter option: 6 Success: users sorted Press ENTER to continue

======================================== | Option: Print all users | ======================================== ================================================================================================================================= | ID | Full Name | Age | Profession | Friends Rating | Public Rating | Friends count | Friends IDs | |----|------------------------|-----|------------------|----------------|---------------|---------------|-----------------------| | 1 | John Doe | 10 | undefined | 4.5 | 3.9 | 1 | 2 | | 12 | Yakui The Maid | 35 | musician | 5.0 | 4.0 | 1 | 9 | | 2 | Jane Doe | 20 | undefined | 4.0 | 4.0 | 1 | 1 | | 13 | God is an Astronaut | 20 | musician | 5.0 | 5.0 | 2 | 1, 12 | | 11 | David Wilson | 35 | actor | 4.0 | 4.2 | 2 | 5, 2 | | 8 | Jane Smith | 25 | driver | 3.8 | 4.1 | 2 | 1, 3 | | 10 | Emily Davis | 27 | driver | 4.1 | 3.8 | 3 | 1, 2, 3 | | 5 | Robert White | 29 | dentist | 4.3 | 3.8 | 3 | 1, 2, 3 | | 14 | new user | 0 | undefined | 2.0 | 0.0 | 4 | 1, 2, 3, 5 | | 7 | Linda Martinez | 32 | pilot | 3.9 | 3.7 | 4 | 4, 6, 5, 1 | | 3 | Alice Johnson | 30 | pilot | 4.2 | 3.7 | 4 | 1, 2, 6, 8 | | 6 | Michael Brown | 33 | engineer | 3.9 | 4.0 | 5 | 3, 6, 9, 10, 2 | | 4 | Sarah Taylor | 31 | teacher | 4.0 | 4.1 | 5 | 8, 5, 6, 3, 1 | | 9 | Jack London | 31 | writer | 5.0 | 5.0 | 6 | 8, 5, 6, 3, 1, 9 | ================================================================================================================================= Press ENTER to continue

======================================== | Option: Delete profession | ======================================== ======================================== | ID | Name | |----|---------------------------------| | 1 | pilot | | 2 | engineer | | 3 | teacher | | 4 | driver | | 5 | dentist | | 6 | actor | | 7 | writer | | 8 | musician | | 9 | test long name of new profes... | ======================================== Enter profession id to delete profession before it (or 0 to return to menu): 3 Profession with id 3: ======================================== | ID | Name | |----|---------------------------------| | 3 | teacher | ======================================== Success: profession with id 3 has been removed! Press ENTER to continue

======================================== | Option: Print all professions | ======================================== ======================================== | ID | Name | |----|---------------------------------| | 1 | pilot | | 2 | engineer | | 4 | driver | | 5 | dentist | | 6 | actor | | 7 | writer | | 8 | musician | | 9 | test long name of new profes... | ======================================== Press ENTER to continue

======================================== | Option: Print all users | ======================================== ================================================================================================================================= | ID | Full Name | Age | Profession | Friends Rating | Public Rating | Friends count | Friends IDs | |----|------------------------|-----|------------------|----------------|---------------|---------------|-----------------------| | 1 | John Doe | 10 | undefined | 4.5 | 3.9 | 1 | 2 | | 12 | Yakui The Maid | 35 | musician | 5.0 | 4.0 | 1 | 9 | | 2 | Jane Doe | 20 | undefined | 4.0 | 4.0 | 1 | 1 | | 13 | God is an Astronaut | 20 | musician | 5.0 | 5.0 | 2 | 1, 12 | | 11 | David Wilson | 35 | actor | 4.0 | 4.2 | 2 | 5, 2 | | 8 | Jane Smith | 25 | driver | 3.8 | 4.1 | 2 | 1, 3 | | 10 | Emily Davis | 27 | driver | 4.1 | 3.8 | 3 | 1, 2, 3 | | 5 | Robert White | 29 | dentist | 4.3 | 3.8 | 3 | 1, 2, 3 | | 14 | new user | 0 | undefined | 2.0 | 0.0 | 4 | 1, 2, 3, 5 | | 7 | Linda Martinez | 32 | pilot | 3.9 | 3.7 | 4 | 4, 6, 5, 1 | | 3 | Alice Johnson | 30 | pilot | 4.2 | 3.7 | 4 | 1, 2, 6, 8 | | 6 | Michael Brown | 33 | engineer | 3.9 | 4.0 | 5 | 3, 6, 9, 10, 2 | | 4 | Sarah Taylor | 31 | undefined | 4.0 | 4.1 | 5 | 8, 5, 6, 3, 1 | | 9 | Jack London | 31 | writer | 5.0 | 5.0 | 6 | 8, 5, 6, 3, 1, 9 | ================================================================================================================================= Press ENTER to continue

======================================== | Option: Delete user | ======================================== ================================================================================================================================= | ID | Full Name | Age | Profession | Friends Rating | Public Rating | Friends count | Friends IDs | |----|------------------------|-----|------------------|----------------|---------------|---------------|-----------------------| | 1 | John Doe | 10 | undefined | 4.5 | 3.9 | 1 | 2 | | 12 | Yakui The Maid | 35 | musician | 5.0 | 4.0 | 1 | 9 | | 2 | Jane Doe | 20 | undefined | 4.0 | 4.0 | 1 | 1 | | 13 | God is an Astronaut | 20 | musician | 5.0 | 5.0 | 2 | 1, 12 | | 11 | David Wilson | 35 | actor | 4.0 | 4.2 | 2 | 5, 2 | | 8 | Jane Smith | 25 | driver | 3.8 | 4.1 | 2 | 1, 3 | | 10 | Emily Davis | 27 | driver | 4.1 | 3.8 | 3 | 1, 2, 3 | | 5 | Robert White | 29 | dentist | 4.3 | 3.8 | 3 | 1, 2, 3 | | 14 | new user | 0 | undefined | 2.0 | 0.0 | 4 | 1, 2, 3, 5 | | 7 | Linda Martinez | 32 | pilot | 3.9 | 3.7 | 4 | 4, 6, 5, 1 | | 3 | Alice Johnson | 30 | pilot | 4.2 | 3.7 | 4 | 1, 2, 6, 8 | | 6 | Michael Brown | 33 | engineer | 3.9 | 4.0 | 5 | 3, 6, 9, 10, 2 | | 4 | Sarah Taylor | 31 | undefined | 4.0 | 4.1 | 5 | 8, 5, 6, 3, 1 | | 9 | Jack London | 31 | writer | 5.0 | 5.0 | 6 | 8, 5, 6, 3, 1, 9 | ================================================================================================================================= Enter user id to delete user (or 0 to return to menu): 145 Failed: there is no user with id 145 Press ENTER to continue

======================================== | Option: Clear profession list | ======================================== Success: list cleared! Press ENTER to continue

======================================== | Option: Print all professions | ======================================== ======================================== | ID | Name | |----|---------------------------------| ======================================== Press ENTER to continue

======================================== | Option: Print all users | ======================================== ================================================================================================================================= | ID | Full Name | Age | Profession | Friends Rating | Public Rating | Friends count | Friends IDs | |----|------------------------|-----|------------------|----------------|---------------|---------------|-----------------------| | 1 | John Doe | 10 | undefined | 4.5 | 3.9 | 1 | 2 | | 12 | Yakui The Maid | 35 | undefined | 5.0 | 4.0 | 1 | 9 | | 2 | Jane Doe | 20 | undefined | 4.0 | 4.0 | 1 | 1 | | 13 | God is an Astronaut | 20 | undefined | 5.0 | 5.0 | 2 | 1, 12 | | 11 | David Wilson | 35 | undefined | 4.0 | 4.2 | 2 | 5, 2 | | 8 | Jane Smith | 25 | undefined | 3.8 | 4.1 | 2 | 1, 3 | | 10 | Emily Davis | 27 | undefined | 4.1 | 3.8 | 3 | 1, 2, 3 | | 5 | Robert White | 29 | undefined | 4.3 | 3.8 | 3 | 1, 2, 3 | | 14 | new user | 0 | undefined | 2.0 | 0.0 | 4 | 1, 2, 3, 5 | | 7 | Linda Martinez | 32 | undefined | 3.9 | 3.7 | 4 | 4, 6, 5, 1 | | 3 | Alice Johnson | 30 | undefined | 4.2 | 3.7 | 4 | 1, 2, 6, 8 | | 6 | Michael Brown | 33 | undefined | 3.9 | 4.0 | 5 | 3, 6, 9, 10, 2 | | 4 | Sarah Taylor | 31 | undefined | 4.0 | 4.1 | 5 | 8, 5, 6, 3, 1 | | 9 | Jack London | 31 | undefined | 5.0 | 5.0 | 6 | 8, 5, 6, 3, 1, 9 | ================================================================================================================================= Press ENTER to continue

======================================== | Choose an option | |--------------------------------------| | 0. Exit | | 1. Print all users | | 2. Print all professions | | 3. Add new profession | | 4. Add new user | | 5. Update user data | | 6. Filter users | | 7. Sort users | | 8. Delete profession | | 9. Delete user | | 10. Clear user list | | 11. Clear profession list | ======================================== Option: 0 Do you want to save changes? (1 - yes, 0 - no): 0 Bye! Press ENTER to continue

**Текст программы:**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <time.h>

#include <ctype.h>

#define MAXLEN 256

typedef struct professionStruct {

int id;

char name[MAXLEN];

struct professionStruct\* next;

struct professionStruct\* prev;

} Profession;

typedef struct professionHeadStruct {

Profession\* first;

Profession\* last;

int count;

} ProfessionHead;

typedef struct userStruct {

int id;

char \*fullName;

int age;

float friendsRating;

float publicRating;

int friendsCount;

int\* friendsId;

Profession\* profession;

struct userStruct\* next;

struct userStruct\* prev;

} User;

typedef struct userHeadStruct {

User\* first;

User\* last;

int count;

} UserHead;

void printMenu();

void printProfessionHeader();

void printAllProfessions(ProfessionHead\* head);

void printUserHeader();

void printAllUsers(UserHead\* uHead);

void printOptionHeader(const char\* optionDescription);

void pressEnterToContinue();

void clearConsole();

void trimForDisplay(char \*output, const char \*input, int maxLength);

void printUser(User \*user);

void printProfession(Profession \*profession);

void printLongLine();

void printShortLine();

ProfessionHead\* makeProfessionHead();

Profession\* makeProfessionNode(char name[MAXLEN]);

void pushBackProfessionNode(ProfessionHead\* head, Profession\* profession);

void deleteProfessionNode(ProfessionHead\* pHead, UserHead\* uHead, Profession\* profession);

void freeProfessionList(ProfessionHead\* head);

void readProfessions(char\* filename, ProfessionHead\* head);

Profession\* findProfessionById(ProfessionHead\* head, int id);

Profession\* findProfessionByName(ProfessionHead\* head, char name[MAXLEN]);

void writeProfessionsToFile(ProfessionHead\* head, const char\* filename);

UserHead\* makeUserHead();

User\* makeUserNode();

void fillUserNode(ProfessionHead\* pHead, UserHead\* uHead, User\* user, char\*\* str);

void pushBackUserNode(UserHead\* head, User\* user);

void freeUserStruct(User\* user);

void freeUserList(UserHead\* head);

void clearUsersProfessionById(UserHead\* head, int id);

void readUsers(char\* filename, UserHead\* head, ProfessionHead\* pHead);

User\* findUserById(UserHead\* head, int id);

void filterUsersByPublicRating(UserHead\* uHead, float minRating, float maxRating);

void filterUsersByFriendsRating(UserHead\* uHead, float minRating, float maxRating);

void filterUsersByAge(UserHead\* uHead, int minAge, int maxAge);

void filterUsersByFriendsCount(UserHead\* uHead, int minCount, int maxCount);

void filterUsersByProfessionName(UserHead\* uHead, char\* professionName);

void filterUsersByName(UserHead\* uHead, char\* name);

void deleteUserNode(UserHead\* head, User\* user);

int compareUsers(User\* a, User\* b, int option);

void sortUsersByField(UserHead\* uHead, int option);

void writeUsersToFile(UserHead\* head, const char\* filename);

void nullString(char str[MAXLEN]);

void trim(char str[MAXLEN]);

char \*\*split(char \*str, int length, char sep);

void inputIntArray(UserHead\* uHead, User\* user, char \*str, char sep, int isManual);

void getUsersIdList(UserHead\* uHead, int\* dest);

int cmp(const void \*a, const void \*b);

int binarySearch(const int arr[], int start, int end, int target);

int startsWithIgnoreCase(const char \*str, const char \*prefix);

void clearStdin();

void makeLog(const char\* title, const char\* funcName, const char\* log);

void appGUI(ProfessionHead\* pHead, UserHead\* uHead);

void appOption(ProfessionHead\* professionHead, UserHead\* userHead, int option);

void deleteProfessionGUI(ProfessionHead\* head, UserHead\* userHead);

void addProfessionGUI(ProfessionHead\* head);

void specifyUserNameGUI(User\* user);

void specifyUserAgeGUI(User\* user);

void specifyUserFriendsRatingGUI(User\* user);

void specifyUserPublicRatingGUI(User\* user);

void specifyUserProfessionGUI(ProfessionHead\* pHead, User\* user);

void specifyUserFriendsGUI(UserHead\* uHead, User\* user);

void updateUserDataGUI(ProfessionHead\* pHead, UserHead\* uHead);

void addUserGUI(ProfessionHead\* pHead, UserHead\* uHead);

void filterUsersByFieldGUI(UserHead\* uHead);

void deleteUserGUI(UserHead\* head);

void clearProfessionListGUI(ProfessionHead\* pHead, UserHead\* uHead);

void sortUsersByFieldGUI(UserHead\* uHead);

void clearUserListGUI(UserHead\* head);

int main() {

UserHead\* userHead = NULL;

ProfessionHead\* professionHead = NULL;

makeLog("APP START", "main", "App started work");

userHead = makeUserHead();

professionHead = makeProfessionHead();

if (userHead != NULL && professionHead != NULL) {

appGUI(professionHead, userHead);

} else {

printf("Error: memory allocation error\n");

}

makeLog("APP FINISH", "main", "App finished work\n");

return 0;

}

void printMenu() {

printShortLine();

printf("| Choose an option |\n");

printf("|--------------------------------------|\n");

printf("| 0. Exit |\n");

printf("| 1. Print all users |\n");

printf("| 2. Print all professions |\n");

printf("| 3. Add new profession |\n");

printf("| 4. Add new user |\n");

printf("| 5. Update user data |\n");

printf("| 6. Filter users |\n");

printf("| 7. Sort users |\n");

printf("| 8. Delete profession |\n");

printf("| 9. Delete user |\n");

printf("| 10. Clear user list |\n");

printf("| 11. Clear profession list |\n");

printShortLine();

printf("Option: ");

}

void printProfessionHeader() {

printShortLine();

printf("| ID | Name |\n");

printf("|----|---------------------------------|\n");

}

void printAllProfessions(ProfessionHead\* head) {

Profession \*q;

printProfessionHeader();

q = head->first;

while (q != NULL) {

printProfession(q);

q = q->next;

}

printShortLine();

}

void printUserHeader() {

printLongLine();

printf("| ID | Full Name | Age | Profession | Friends Rating | Public Rating | Friends count | Friends IDs |\n");

printf("|----|------------------------|-----|------------------|----------------|---------------|---------------|-----------------------|\n");

}

void printAllUsers(UserHead\* uHead) {

User \*q;

printUserHeader();

q = uHead->first;

while (q != NULL) {

printUser(q);

q = q->next;

}

printLongLine();

}

void printLongLine() {

printf("=================================================================================================================================\n");

}

void printShortLine() {

printf("========================================\n");

}

void printOptionHeader(const char\* optionDescription) {

printShortLine();

printf("| Option: %-28s |\n", optionDescription);

printShortLine();

printf("\n");

}

void pressEnterToContinue() {

printf("\nPress ENTER to continue ");

clearStdin();

clearConsole();

}

void clearConsole() {

#if defined(\_WIN32) || defined(\_WIN64)

system("cls");

#else

system("clear");

#endif

}

void trimForDisplay(char \*output, const char \*input, int maxLength) {

if (strlen(input) > maxLength) {

strncpy(output, input, maxLength - 3);

output[maxLength - 3] = '\0';

strcat(output, "...");

} else {

strcpy(output, input);

}

}

void printUser(User \*user) {

char friendsIds[MAXLEN] = "";

char idStr[10];

int i;

char profession[MAXLEN] = "undefined";

char trimmedFullName[23], trimmedProfession[17], trimmedFriendsIds[30];

if (user->profession != NULL) {

trimForDisplay(profession, user->profession->name, sizeof(profession));

}

if (user->friendsId != NULL) {

for (i = 0; i < user->friendsCount; i++) {

sprintf(idStr, "%d", user->friendsId[i]);

strcat(friendsIds, idStr);

if (i < user->friendsCount - 1) {

strcat(friendsIds, ", ");

}

}

}

trimForDisplay(trimmedFullName, user->fullName, 22);

trimForDisplay(trimmedProfession, profession, 16);

trimForDisplay(trimmedFriendsIds, friendsIds, 21);

printf("| %-2d | %-22s | %-3d | %-16s | %-14.1f | %-13.1f | %-13d | %21s |\n",

user->id, trimmedFullName, user->age, trimmedProfession, user->friendsRating, user->publicRating, user->friendsCount, trimmedFriendsIds);

}

void printProfession(Profession \*profession) {

char trimmedProfessionName[32];

trimForDisplay(trimmedProfessionName, profession->name, 31);

printf("| %-2d | %-31s |\n", profession->id, trimmedProfessionName);

}

ProfessionHead\* makeProfessionHead() {

ProfessionHead\* head = NULL;

head = (ProfessionHead\*)malloc(sizeof(ProfessionHead));

if (head != NULL) {

head->count = 0;

head->first = NULL;

head->last = NULL;

} else {

perror("Memory allocation failed");

makeLog("ERROR", "makeProfessionHead", "Memory allocation failed (head)");

}

return head;

}

Profession\* makeProfessionNode(char name[MAXLEN]) {

Profession\* profession = NULL;

profession = (Profession\*)malloc(sizeof(Profession));

if (profession != NULL) {

profession->id = 0;

strcpy(profession->name, name);

profession->next = NULL;

profession->prev = NULL;

}

return profession;

}

void pushBackProfessionNode(ProfessionHead\* head, Profession\* profession) {

head->count++;

if (head->first == NULL) { /\* list is empty \*/

head->first = profession; /\* first element is profession \*/

head->last = profession; /\* last element is profession \*/

profession->id = 1;

} else { /\* list has only one element \*/

profession->id = head->last->id + 1;

profession->prev = head->last; /\* profession's previous element is last element \*/

head->last->next = profession; /\* profession becomes element after last element \*/

head->last = profession; /\* profession becomes last element \*/

}

}

void deleteProfessionNode(ProfessionHead\* pHead, UserHead\* uHead, Profession\* profession) {

if (pHead->first == profession) {

pHead->first = profession->next;

if (profession->next != NULL) {

profession->next->prev = profession->prev;

}

} else if (pHead->last == profession) {

pHead->last = profession->prev;

if (profession->prev != NULL) {

profession->prev->next = profession->next;

}

} else {

if (profession->prev != NULL) {

profession->prev->next = profession->next;

}

if (profession->next != NULL) {

profession->next->prev = profession->prev;

}

}

clearUsersProfessionById(uHead, profession->id);

free(profession);

pHead->count--;

}

void freeProfessionList(ProfessionHead\* head) {

Profession \*q, \*q1;

q = head->first;

while (q != NULL) {

q1 = q->next;

free(q);

q = q1;

}

free(head);

}

void readProfessions(char\* filename, ProfessionHead\* head) {

FILE\* file;

Profession\* profession;

int n, count, i;

char temp[MAXLEN];

profession = NULL;

n = count = 0;

file = fopen(filename, "r");

if (file != NULL) {

makeLog("FILE READ", "readProfessions", filename);

while ((fgets(temp, MAXLEN, file)) != NULL) n++;

rewind(file);

for (i = 0; i < n; i++) {

nullString(temp);

fgets(temp, MAXLEN, file);

trim(temp);

profession = makeProfessionNode(temp);

if (profession != NULL) {

pushBackProfessionNode(head, profession);

count++;

}

}

fclose(file);

} else {

perror("Failed to open file");

makeLog("ERROR", "readProfessions", "Failed to open file");

}

if (count != n) {

perror("Failed to read from file");

freeProfessionList(head);

}

}

Profession\* findProfessionByName(ProfessionHead\* head, char name[MAXLEN]) {

Profession\* q = NULL;

q = head->first;

while (q != NULL && strcmp(q->name, name) != 0) {

q = q->next;

}

return q;

}

Profession\* findProfessionById(ProfessionHead\* head, int id) {

Profession\* q = NULL;

q = head->first;

while (q != NULL && q->id != id) {

q = q->next;

}

return q;

}

void writeProfessionsToFile(ProfessionHead\* head, const char\* filename) {

FILE\* file = fopen(filename, "w");

Profession\* current = NULL;

if (file != NULL) {

makeLog("FILE WRITE", "writeProfessionsToFile", filename);

current = head->first;

while (current != NULL) {

fprintf(file, "%s\n", current->name);

current = current->next;

}

fclose(file);

} else {

makeLog("ERROR", "writeProfessionsToFile", "Failed to open file");

perror("Failed to open file");

}

}

UserHead\* makeUserHead() {

UserHead\* head = NULL;

head = (UserHead\*)malloc(sizeof(UserHead));

if (head != NULL) {

head->count = 0;

head->first = NULL;

head->last = NULL;

} else {

perror("Memory allocation failed");

makeLog("ERROR", "makeUserHead", "Memory allocation failed (head)");

}

return head;

}

User\* makeUserNode() {

User\* user = NULL;

user = (User\*)malloc(sizeof(User));

if (user != NULL) {

user->age = 0;

user->friendsCount = 0;

user->publicRating = 0;

user->friendsRating = 0;

user->id = 0;

user->fullName = NULL;

user->profession = NULL;

user->friendsId = NULL;

user->next = NULL;

user->prev = NULL;

user->id = 0;

}

return user;

}

void fillUserNode(ProfessionHead\* pHead, UserHead\* uHead, User\* user, char\*\* str) {

if (user != NULL) {

user->fullName = str[0];

user->age = atoi(str[1]);

free(str[1]);

user->profession = findProfessionByName(pHead, str[2]);

free(str[2]);

user->friendsRating = atof(str[3]);

free(str[3]);

user->publicRating = atof(str[4]);

free(str[4]);

user->friendsCount = atoi(str[5]);

free(str[5]);

if (user->friendsCount > 0) {

user->friendsId = NULL;

inputIntArray(uHead, user, str[6], ',', 0);

} else {

user->friendsId = NULL;

}

free(str[6]);

free(str);

user->next = NULL;

user->prev = NULL;

} else {

perror("Memory allocation failed");

makeLog("ERROR", "makeUserNode", "Memory allocation failed (user)");

}

}

void pushBackUserNode(UserHead\* head, User\* user) {

head->count++;

if (head->first == NULL) {

head->first = user;

head->last = user;

user->id = 1;

} else {

user->id = head->last->id + 1;

user->prev = head->last;

head->last->next = user;

head->last = user;

}

}

void freeUserStruct(User\* user) {

if (user->fullName != NULL) {

free(user->fullName);

user->fullName = NULL;

}

if (user->friendsId != NULL) {

free(user->friendsId);

user->friendsId = NULL;

}

if (user->profession != NULL) {

user->profession = NULL;

}

free(user);

}

void freeUserList(UserHead\* head) {

User \*q = NULL, \*q1 = NULL;

/\* char buffer[MAXLEN]; \*/

q = head->first;

/\* makeLog("LIST FREE", "freeUserList", "start"); \*/

while (q != NULL) {

/\* sprintf(buffer, "%p", q->next); \*/

/\* makeLog("attempt to get q->next", "freeUserList", buffer); \*/

q1 = q->next;

freeUserStruct(q);

q = q1;

}

/\* sprintf(buffer, "%p", head); \*/

/\* makeLog("attempt to free head", "freeUserList", buffer); \*/

free(head);

}

void clearUsersProfessionById(UserHead\* head, int id) {

User\* q = NULL;

q = head->first;

while (q != NULL) {

if (q->profession != NULL && q->profession->id == id) {

q->profession = NULL;

}

q = q->next;

}

}

void readUsers(char\* filename, UserHead\* head, ProfessionHead\* pHead) {

FILE\* file;

User\* user;

int n, count, i, slen;

char\*\* splitArray;

char temp[MAXLEN];

user = NULL;

n = count = 0;

file = fopen(filename, "r");

if (file != NULL) {

makeLog("FILE READ", "readUsers", filename);

while ((fgets(temp, MAXLEN, file)) != NULL) n++;

rewind(file);

for (i = 0; i < n; i++, count++) {

nullString(temp);

fgets(temp, MAXLEN, file);

slen = strlen(temp);

trim(temp);

splitArray = split(temp, slen, ';');

if (splitArray != NULL) {

user = makeUserNode();

if (user != NULL) {

fillUserNode(pHead, head, user, splitArray);

pushBackUserNode(head, user);

}

}

}

fclose(file);

} else {

perror("Failed to open file");

makeLog("ERROR", "readUsers", "Failed to open file");

}

if (count != n) {

perror("Failed to read from file");

freeUserList(head);

}

}

User\* findUserById(UserHead\* head, int id) {

User\* q = NULL;

q = head->first;

while (q != NULL && q->id != id) {

q = q->next;

}

return q;

}

void filterUsersByName(UserHead\* uHead, char\* name) {

User \*q;

printUserHeader();

q = uHead->first;

while (q != NULL) {

if (startsWithIgnoreCase(q->fullName, name) == 1) {

printUser(q);

}

q = q->next;

}

printLongLine();

}

void filterUsersByProfessionName(UserHead\* uHead, char\* professionName) {

User \*q;

printUserHeader();

q = uHead->first;

while (q != NULL) {

if ((q->profession != NULL && startsWithIgnoreCase(q->profession->name, professionName) == 1) || (q->profession == NULL && startsWithIgnoreCase("undefined", professionName) == 1)) {

printUser(q);

}

q = q->next;

}

printLongLine();

}

void filterUsersByAge(UserHead\* uHead, int minAge, int maxAge) {

User \*q;

printUserHeader();

q = uHead->first;

while (q != NULL) {

if (q->age >= minAge && q->age <= maxAge) {

printUser(q);

}

q = q->next;

}

printLongLine();

}

void filterUsersByFriendsRating(UserHead\* uHead, float minRating, float maxRating) {

User \*q;

printUserHeader();

q = uHead->first;

while (q != NULL) {

if (q->friendsRating >= minRating && q->friendsRating <= maxRating) {

printUser(q);

}

q = q->next;

}

printLongLine();

}

void filterUsersByPublicRating(UserHead\* uHead, float minRating, float maxRating) {

User \*q;

printUserHeader();

q = uHead->first;

while (q != NULL) {

if (q->publicRating >= minRating && q->publicRating <= maxRating) {

printUser(q);

}

q = q->next;

}

printLongLine();

}

void filterUsersByFriendsCount(UserHead\* uHead, int minCount, int maxCount) {

User \*q;

printUserHeader();

q = uHead->first;

while (q != NULL) {

if (q->friendsCount >= minCount && q->friendsCount <= maxCount) {

printUser(q);

}

q = q->next;

}

printLongLine();

}

void deleteUserNode(UserHead\* head, User\* user) {

User\* q = NULL;

int\* tempPtr;

int i, j, check;

int temp[MAXLEN] = {0};

q = head->first;

while (q != NULL) {

if (q->friendsCount > 0 && q->friendsId != NULL) {

check = 0;

j = 0;

for (i = 0; i < q->friendsCount; i++) {

if (q->friendsId[i] != user->id) {

temp[j++] = q->friendsId[i];

} else {

check = 1;

}

}

if (check) {

q->friendsCount--;

if (q->friendsCount != 0) {

tempPtr = (int\*)malloc(q->friendsCount \* sizeof(int));

if (tempPtr != NULL) {

free(q->friendsId);

q->friendsId = tempPtr;

for (i = 0; i < q->friendsCount; i++) {

q->friendsId[i] = temp[i];

}

} else {

perror("Memory allocation failed");

}

} else {

free(q->friendsId);

q->friendsId = NULL;

}

}

}

q = q->next;

}

if (head->first == user) {

head->first = user->next;

if (user->next != NULL) {

user->next->prev = user->prev;

}

} else if (head->last == user) {

head->last = user->prev;

if (user->prev != NULL) {

user->prev->next = user->next;

}

} else {

if (user->prev != NULL) {

user->prev->next = user->next;

}

if (user->next != NULL) {

user->next->prev = user->prev;

}

}

freeUserStruct(user);

head->count--;

}

void sortUsersByField(UserHead\* head, int option) {

User\* sorted = NULL;

User\* current = head->first;

User\* next = NULL;

User\* temp = NULL;

if (head->first != NULL && head->first->next != NULL) {

while (current != NULL) {

next = current->next;

if (sorted == NULL || compareUsers(current, sorted, option) < 0) {

current->next = sorted;

if (sorted != NULL) sorted->prev = current;

sorted = current;

sorted->prev = NULL;

} else {

temp = sorted;

while (temp->next != NULL && compareUsers(current, temp->next, option) > 0) {

temp = temp->next;

}

current->next = temp->next;

if (temp->next != NULL) temp->next->prev = current;

temp->next = current;

current->prev = temp;

}

current = next;

}

head->first = sorted;

temp = sorted;

while (temp != NULL && temp->next != NULL) {

temp = temp->next;

}

head->last = temp;

}

}

int compareUsers(User\* a, User\* b, int option) {

int result;

switch (option) {

case 1:

result = a->id - b->id;

break;

case 2:

result = strcmp(a->fullName, b->fullName);

break;

case 3:

result = a->age - b->age;

break;

case 4:

result = (a->friendsRating > b->friendsRating) ? 1 : (a->friendsRating < b->friendsRating) ? -1 : 0;

break;

case 5:

result = (a->publicRating > b->publicRating) ? 1 : (a->publicRating < b->publicRating) ? -1 : 0;

break;

case 6:

result = a->friendsCount - b->friendsCount;

break;

default:

result = 0;

break;

}

return result;

}

void writeUsersToFile(UserHead\* head, const char\* filename) {

FILE\* file = fopen(filename, "w");

User\* current = NULL;

char\* professionName;

int i;

if (file != NULL) {

makeLog("FILE WRITE", "writeUsersToFile", filename);

current = head->first;

while (current != NULL) {

professionName = "undefined";

if (current->profession != NULL) {

professionName = current->profession->name;

}

fprintf(file, "%s;%d;%s;%.1f;%.1f;%d", current->fullName, current->age, professionName,

current->friendsRating, current->publicRating, current->friendsCount);

if (current->friendsCount > 0 && current->friendsId != NULL) {

fprintf(file, ";");

for (i = 0; i < current->friendsCount; i++) {

fprintf(file, "%d", current->friendsId[i]);

if (i < current->friendsCount - 1) {

fprintf(file, ",");

}

}

}

fprintf(file, "\n");

current = current->next;

}

fclose(file);

} else {

printf("Failed to open file %s\n", filename);

makeLog("ERROR", "writeUsersToFile", "Failed to open file");

}

}

void nullString(char str[MAXLEN]) {

int i;

for (i = 0; i < MAXLEN; i++) {

str[i] = '\0';

}

}

void trim(char str[MAXLEN]) {

int i, flag = 0;

str[MAXLEN - 1] = '\0';

for (i = 0; str[i] != '\0' && !flag; i++) {

if (str[i] == '\n' || str[i] == '\r') {

str[i] = '\0';

flag = 1;

}

}

}

char \*\*split(char \*str, int length, char sep) {

int count = 0;

int i = 0;

int start = 0;

int j = 0;

int wordLen = 0;

char \*\*result = NULL;

char \*newStr = NULL;

int allocError = 0;

for (i = 0; i < length; i++) {

if (str[i] == sep) count++;

}

count++;

result = malloc(count \* sizeof(char \*));

if (result == NULL) {

perror("Memory allocation failed");

makeLog("ERROR", "split", "Memory allocation failed (result)");

} else {

for (i = 0; i < length; i++) {

if (str[i] == ';' || str[i] == '\0') {

wordLen = i - start;

newStr = malloc((wordLen + 1) \* sizeof(char));

if (newStr == NULL) {

perror("Memory allocation failed");

allocError = 1;

i = length;

} else {

strncpy(newStr, str + start, wordLen);

newStr[wordLen] = '\0';

result[j++] = newStr;

start = i + 1;

}

}

}

if (allocError) {

for (i = 0; i < j; i++) {

free(result[i]);

}

free(result);

result = NULL;

}

}

return result;

}

void inputIntArray(UserHead\* uHead, User\* user, char \*str, char sep, int isManual) {

int enteredIdCount = 0, sepCount = 0, unicIdCount = 0, actualIdCount = 0, startIndex, foundIndex;

int start = 0;

int i, len, isInputValid, n;

char tempStr[MAXLEN] = {0};

int enteredIds[MAXLEN] = {0};

int unicEnteredIds[MAXLEN] = {0};

int actualIds[MAXLEN] = {0};

int idList[MAXLEN] = {0};

if (strlen(str) != 0) {

for (i = 0; str[i] != '\0'; i++) {

if (str[i] == sep) sepCount++;

}

sepCount++;

if (sepCount > MAXLEN) {

printf("It seems that the number of entered IDs is too big -> updating friends count: %d\n", MAXLEN);

sepCount = MAXLEN - 1;

}

if (user->friendsCount != sepCount) {

printf("It seems that the number of entered IDs does not correspond to the specified number of friends\n");

if (sepCount < uHead->count) {

user->friendsCount = sepCount;

} else {

user->friendsCount = uHead->count;

}

printf("updating friends count: %d\n", user->friendsCount);

}

isInputValid = 1;

for (i = 0; str[i] != '\0' && isInputValid && enteredIdCount < sepCount; i++) {

if (str[i] == ',' || str[i + 1] == '\0') {

len = (str[i] == ',') ? (i - start) : (i - start + 1);

strncpy(tempStr, str + start, len);

tempStr[len] = '\0';

n = atoi(tempStr);

if (n != 0) {

enteredIds[enteredIdCount++] = n;

start = i + 1;

} else {

printf("It seems that your input is not valid. Please check your input and try again\n");

isInputValid = 0;

}

}

}

if (!isManual) {

user->friendsId = malloc(enteredIdCount \* sizeof(int));

user->friendsCount = enteredIdCount;

if (user->friendsId == NULL) {

perror("Memory allocation failed");

} else {

for (i = 0; i < enteredIdCount; i++) {

user->friendsId[i] = enteredIds[i];

}

}

}

if (!isInputValid) {

user->friendsCount = 0;

}

if (isInputValid && isManual) {

getUsersIdList(uHead, idList);

qsort(idList, uHead->count, sizeof(int), cmp);

qsort(enteredIds, enteredIdCount, sizeof(int), cmp);

unicIdCount = 1;

unicEnteredIds[0] = enteredIds[0];

for (i = 1; i < enteredIdCount; i++) {

if (enteredIds[i] != enteredIds[i - 1]) {

unicEnteredIds[unicIdCount++] = enteredIds[i];

} else {

printf("Duplicated ID: %d\n", enteredIds[i]);

}

}

if (unicIdCount != user->friendsCount) {

printf("It seems that some IDs are entered more than once -> updating friends count: %d\n", unicIdCount);

user->friendsCount = unicIdCount;

}

startIndex = 0;

actualIdCount = 0;

for (i = 0; i < unicIdCount; i++) {

foundIndex = binarySearch(idList, startIndex, uHead->count - 1, unicEnteredIds[i]);

if (foundIndex != -1) {

startIndex = foundIndex;

actualIds[actualIdCount++] = unicEnteredIds[i];

} else {

printf("ID not found: %d\n", unicEnteredIds[i]);

}

}

if (actualIdCount != unicIdCount) {

printf("It seems that list of users does not contain some of entered IDs -> updating friends count: %d\n", actualIdCount);

user->friendsCount = actualIdCount;

}

if (user->friendsId != NULL) {

free(user->friendsId);

}

user->friendsId = malloc(actualIdCount \* sizeof(int));

if (user->friendsId == NULL) {

perror("Memory allocation failed");

makeLog("ERROR", "inpuIntArray", "Memory allocation failed (user->friendsId)");

} else {

for (i = 0; i < actualIdCount; i++) {

user->friendsId[i] = actualIds[i];

}

printf("Success: friends ids specified\n\n");

}

}

} else {

user->friendsCount = 0;

printf("Seems that your user does not have any friends\n");

}

}

void getUsersIdList(UserHead\* uHead, int\* dest) {

User\* tempUser = uHead->first;

int i = 0;

while (tempUser != NULL) {

dest[i++] = tempUser->id;

tempUser = tempUser->next;

}

}

int cmp(const void \*a, const void \*b) {

return (\*(int\*)a - \*(int\*)b);

}

int binarySearch(const int arr[], int start, int end, int target) {

int result, isFound, mid;

result = -1;

isFound = 0;

while (start <= end && !isFound) {

mid = start + (end - start) / 2;

if (arr[mid] == target) {

isFound = 1;

result = mid;

} else if (arr[mid] < target) {

start = mid + 1;

} else {

end = mid - 1;

}

}

return result;

}

int startsWithIgnoreCase(const char \*str, const char \*prefix) {

int isPrefix = 1;

while (\*str && \*prefix && isPrefix) {

if (tolower(\*str) != tolower(\*prefix)) {

isPrefix = 0;

}

str++;

prefix++;

}

if (\*prefix != '\0') {

isPrefix = 0;

}

return isPrefix;

}

void clearStdin() {

int c;

while ((c = getchar()) != '\n' && c != EOF) { }

}

void makeLog(const char\* title, const char\* funcName, const char\* log) {

FILE\* file = fopen("program.log", "a");

struct tm\* timeinfo;

char timeStr[80];

time\_t rawtime;

if (file == NULL) {

perror("Error opening log file");

} else {

time(&rawtime);

timeinfo = localtime(&rawtime);

strftime(timeStr, sizeof(timeStr), "%Y-%m-%dT%H:%M:%S", timeinfo);

fprintf(file, "%-19s | FROM %-30s: %-15s %s\n", timeStr, funcName, title, log);

fclose(file);

}

}

void appGUI(ProfessionHead\* professionHead, UserHead\* userHead) {

int option, doYouWantToSave;

readProfessions("professions.csv", professionHead);

readUsers("users.csv", userHead, professionHead);

do {

clearConsole();

printMenu();

scanf("%d", &option);

clearStdin();

if (option != 0) {

appOption(professionHead, userHead, option);

} else {

doYouWantToSave = -1;

printf("\nDo you want to save changes? (1 - yes, 0 - no): ");

scanf("%d", &doYouWantToSave);

clearStdin();

if (doYouWantToSave == 1) {

writeUsersToFile(userHead, "users.csv");

writeProfessionsToFile(professionHead, "professions.csv");

printf("\nSuccess: changes saved!\n");

printf("\nBye!\n");

} else if (doYouWantToSave != 0) {

option = -1;

printf("\nFailed: option must be 0 or 1\n");

} else if (doYouWantToSave == 0) {

printf("\nBye!\n");

}

pressEnterToContinue();

clearConsole();

}

} while (option != 0);

freeProfessionList(professionHead);

freeUserList(userHead);

}

void appOption(ProfessionHead\* professionHead, UserHead\* userHead, int option) {

clearConsole();

switch (option) {

case 1:

printOptionHeader("Print all users");

printAllUsers(userHead);

break;

case 2:

printOptionHeader("Print all professions");

printAllProfessions(professionHead);

break;

case 3:

printOptionHeader("Add new profession");

addProfessionGUI(professionHead);

break;

case 4:

printOptionHeader("Add new user");

addUserGUI(professionHead, userHead);

break;

case 5:

printOptionHeader("Update user data");

updateUserDataGUI(professionHead, userHead);

break;

case 6:

printOptionHeader("Filter users");

filterUsersByFieldGUI(userHead);

break;

case 7:

printOptionHeader("Sort users");

sortUsersByFieldGUI(userHead);

break;

case 8:

printOptionHeader("Delete profession");

deleteProfessionGUI(professionHead, userHead);

break;

case 9:

printOptionHeader("Delete user");

deleteUserGUI(userHead);

break;

case 10:

printOptionHeader("Clear user list");

clearUserListGUI(userHead);

break;

case 11:

printOptionHeader("Clear profession list");

clearProfessionListGUI(professionHead, userHead);

break;

default:

clearConsole();

printf("\nFailed: invalid option\n");

break;

}

pressEnterToContinue();

}

void deleteProfessionGUI(ProfessionHead\* pHead, UserHead\* uHead) {

int id;

Profession\* profession = NULL;

if (pHead->first != NULL) {

printAllProfessions(pHead);

printf("\nEnter profession id to delete profession before it (or 0 to return to menu): ");

scanf("%d", &id);

clearStdin();

if (id > 0) {

profession = findProfessionById(pHead, id);

if (profession == NULL) {

printf("\nFailed: there is no profession with id %d\n", id);

} else {

printf("\nProfession with id %d:\n", id);

printProfessionHeader();

printProfession(profession);

printShortLine();

deleteProfessionNode(pHead, uHead, profession);

printf("\nSuccess: profession with id %d has been removed!\n", id);

}

} else if (id != 0) {

printf("\nFailed: ID must be always positive\n");

}

} else {

printf("The list of professions is empty\n");

printf("You can add new profession in menu with option 4\n");

}

}

void addUserGUI(ProfessionHead\* pHead, UserHead\* uHead) {

User\* user = NULL;

user = makeUserNode();

if (user != NULL) {

printf("Enter information for new user:\n");

specifyUserNameGUI(user);

specifyUserAgeGUI(user);

specifyUserFriendsRatingGUI(user);

specifyUserPublicRatingGUI(user);

specifyUserFriendsGUI(uHead, user);

specifyUserProfessionGUI(pHead, user);

pushBackUserNode(uHead, user);

printf("\nSuccess: user has been added!\n");

printUserHeader();

printUser(user);

printLongLine();

} else {

makeLog("ERROR", "addUserGUI", "Memory allocation failed (user)");

}

}

void deleteUserGUI(UserHead\* head) {

int id;

User\* user = NULL;

if (head->first != NULL) {

printAllUsers(head);

printf("\nEnter user id to delete user (or 0 to return to menu): ");

scanf("%d", &id);

clearStdin();

if (id > 0) {

user = findUserById(head, id);

if (user == NULL) {

printf("\nFailed: there is no user with id %d\n", id);

} else {

printf("\nUser with id %d:\n", id);

printUserHeader();

printUser(user);

printLongLine();

deleteUserNode(head, user);

printf("\nSuccess: user with id %d has been removed!\n", id);

}

} else if (id != 0) {

printf("\nFailed: ID must be always positive\n");

}

} else {

printf("The list of users is empty\n");

printf("You can add new user in menu with option 0\n");

}

}

void clearProfessionListGUI(ProfessionHead\* pHead, UserHead\* uHead) {

Profession \*q, \*q1;

User\* user;

q = pHead->first;

if (q == NULL) {

printf("There are no profession in the list\n");

} else {

while (q != NULL) {

q1 = q->next;

free(q);

q = q1;

}

user = uHead->first;

while (user != NULL) {

user->profession = NULL;

user = user->next;

}

pHead->first = NULL;

pHead->last = NULL;

pHead->count = 0;

printf("Success: list cleared!\n");

}

}

void clearUserListGUI(UserHead\* head) {

User\* q, \*q1;

if (head->first != NULL) {

q = head->first;

while (q != NULL) {

q1 = q->next;

freeUserStruct(q);

q = q1;

}

head->last = NULL;

head->first = NULL;

head->count = 0;

printf("Success: list cleared!\n");

} else {

printf("The list of users is empty\n");

printf("You can add new user in menu with option 0\n");

}

}

void addProfessionGUI(ProfessionHead\* head) {

char temp[MAXLEN];

Profession\* profession = NULL;

printf("Enter profession name: ");

if (fgets(temp, MAXLEN, stdin) != NULL) {

trim(temp);

profession = makeProfessionNode(temp);

if (profession != NULL) {

pushBackProfessionNode(head, profession);

printf("\nSuccess: profession added\n");

printProfessionHeader();

printProfession(profession);

printShortLine();

} else {

printf("\nFailed: memory error\n");

}

} else {

makeLog("ERROR", "addProfessionGUI", "Memory allocation failed (fgets)");

printf("\nFailed: memory error\n");

}

}

void updateUserDataGUI(ProfessionHead\* pHead, UserHead\* uHead) {

User\* user;

int userId, option;

printAllUsers(uHead);

printf("Enter user id: ");

scanf("%d", &userId);

clearStdin();

user = findUserById(uHead, userId);

if (user != NULL) {

clearConsole();

printOptionHeader("Update user data");

printUserHeader();

printUser(user);

printLongLine();

printf("Which field do you want to edit?\n");

printf("1. full name\n");

printf("2. age\n");

printf("3. profession\n");

printf("4. friends rating\n");

printf("5. public rating\n");

printf("6. friends\n");

printf("7. all fields\n");

printf("Enter option: ");

scanf("%d", &option);

clearStdin();

switch (option) {

case 1:

printOptionHeader("Specify user name");

specifyUserNameGUI(user);

break;

case 2:

printOptionHeader("Specify user age");

specifyUserAgeGUI(user);

break;

case 3:

printOptionHeader("Specify user profession");

specifyUserProfessionGUI(pHead, user);

break;

case 4:

printOptionHeader("Specify user friends rating");

specifyUserFriendsRatingGUI(user);

break;

case 5:

printOptionHeader("Specify user public rating");

specifyUserPublicRatingGUI(user);

break;

case 6:

printOptionHeader("Specify user friends");

specifyUserFriendsGUI(uHead, user);

break;

case 7:

printOptionHeader("Specify all fields");

specifyUserNameGUI(user);

specifyUserAgeGUI(user);

specifyUserProfessionGUI(pHead, user);

specifyUserFriendsRatingGUI(user);

specifyUserPublicRatingGUI(user);

specifyUserFriendsGUI(uHead, user);

break;

default:

printf("\nFailed: wrong option\n");

break;

}

printf("\nUpdated user:\n");

printUserHeader();

printUser(user);

printLongLine();

} else {

printf("\nFailed: user not found\n");

}

}

void specifyUserNameGUI(User\* user) {

char temp[MAXLEN];

printf("Enter user name: ");

if (fgets(temp, MAXLEN, stdin) != NULL) {

trim(temp);

if (user->fullName != NULL) {

free(user->fullName);

user->fullName = NULL;

}

user->fullName = (char\*)malloc(strlen(temp) + 1);

if (user->fullName != NULL) {

strcpy(user->fullName, temp);

printf("Success: name specified\n\n");

} else {

printf("Failed: memory error\n\n");

makeLog("ERROR", "specifyUserNameGUI", "Memory allocation failed (user->fullName)");

}

} else {

makeLog("ERROR", "specifyUserNameGUI", "Memory allocation failed (fgets)");

printf("Failed: memory error\n\n");

}

}

void specifyUserAgeGUI(User\* user) {

int age;

int success;

printf("Enter user age: ");

success = scanf("%d", &age);

clearStdin();

if (age < 0 || age > 200 || success != 1) {

printf("Failed: invalid or impossible age\n\n");

} else {

user->age = age;

printf("Success: age specified\n\n");

}

}

void specifyUserFriendsRatingGUI(User\* user) {

float rating;

int success;

printf("Enter user friends rating: ");

success = scanf("%f", &rating);

clearStdin();

if (rating < 0 || rating > 5 || success != 1) {

printf("Failed: invalid or impossible rating\n\n");

} else {

user->friendsRating = rating;

printf("Success: friends rating specified\n\n");

}

}

void specifyUserPublicRatingGUI(User\* user) {

float rating;

int success;

printf("Enter user public rating: ");

success = scanf("%f", &rating);

clearStdin();

if (rating < 0 || rating > 5 || success != 1) {

printf("Failed: invalid or impossible rating\n\n");

} else {

user->publicRating = rating;

printf("Success: public rating specified\n\n");

}

}

void specifyUserFriendsGUI(UserHead\* uHead, User\* user) {

int friendsCount;

int success;

char temp[MAXLEN];

printf("Enter user friends count (less than %d): ", uHead->count);

success = scanf("%d", &friendsCount);

clearStdin();

if (friendsCount < 0 || friendsCount > uHead->count || success != 1) {

printf("Failed: invalid or impossible friends count\n\n");

} else if (friendsCount == 0) {

user->friendsCount = 0;

if (user->friendsId != NULL) {

free(user->friendsId);

user->friendsId = NULL;

}

printf("Success: friends count specified\n\n");

} else {

printAllUsers(uHead);

user->friendsCount = friendsCount;

printf("Success: friends count specified\n");

printf("Enter user friends ids\n");

printf("Example: 1,2,3,4,5\n");

printf("Enter friends ids: ");

if (fgets(temp, MAXLEN, stdin) != NULL) {

trim(temp);

inputIntArray(uHead, user, temp, ',', 1);

} else {

makeLog("ERROR", "specifyUserFriendsGUI", "Memory allocation failed (fgets)");

printf("Failed: memory error\n\n");

}

}

}

void specifyUserProfessionGUI(ProfessionHead\* pHead, User\* user) {

Profession\* profession;

int success;

int professionId;

if (pHead->first == NULL) {

printf("The list of professions is empty\n");

printf("You can add new profession in menu with option 4\n");

} else {

printAllProfessions(pHead);

printf("Enter profession id: ");

success = scanf("%d", &professionId);

clearStdin();

if (success != 1) {

professionId = 0;

}

profession = findProfessionById(pHead, professionId);

if (profession != NULL) {

user->profession = profession;

printf("Success: profession specified\n\n");

} else {

printf("Failed: profession not found\n\n");

}

}

}

void filterUsersByFieldGUI(UserHead\* uHead) {

int option;

char temp[MAXLEN];

int tempInt;

float tempFloat1, tempFloat2;

printf("1. Name\n");

printf("2. Profession\n");

printf("3. Age\n");

printf("4. Friends Rating\n");

printf("5. Public Rating\n");

printf("6. Friends Count\n");

printf("Enter option: ");

scanf("%d", &option);

clearStdin();

switch (option) {

case 1:

printf("Enter name: ");

if (fgets(temp, MAXLEN, stdin) != NULL) {

trim(temp);

filterUsersByName(uHead, temp);

}

break;

case 2:

printf("Enter profession name: ");

if (fgets(temp, MAXLEN, stdin) != NULL) {

trim(temp);

filterUsersByProfessionName(uHead, temp);

}

break;

case 3:

printf("Enter age: ");

scanf("%d", &tempInt);

clearStdin();

filterUsersByAge(uHead, tempInt, tempInt);

break;

case 4:

printf("Enter min friends rating: ");

scanf("%f", &tempFloat1);

clearStdin();

printf("Enter max friends rating: ");

scanf("%f", &tempFloat2);

clearStdin();

filterUsersByFriendsRating(uHead, tempFloat1, tempFloat2);

break;

case 5:

printf("Enter min public rating: ");

scanf("%f", &tempFloat1);

clearStdin();

printf("Enter max public rating: ");

scanf("%f", &tempFloat2);

clearStdin();

filterUsersByPublicRating(uHead, tempFloat1, tempFloat2);

break;

case 6:

printf("Enter min friends count: ");

scanf("%d", &tempInt);

clearStdin();

printf("Enter max friends count: ");

scanf("%d", &tempInt);

clearStdin();

filterUsersByFriendsCount(uHead, tempInt, tempInt);

break;

default:

printf("Wrong option\n");

break;

}

}

void sortUsersByFieldGUI(UserHead\* uHead) {

int option;

if (uHead->first != NULL) {

printf("1. Sort by id\n");

printf("2. Sort by name\n");

printf("3. Sort by age\n");

printf("4. Sort by friends rating\n");

printf("5. Sort by public rating\n");

printf("6. Sort by friends count\n");

printf("Enter option: ");

scanf("%d", &option);

clearStdin();

if (option > 0 && option <= 6) {

sortUsersByField(uHead, option);

printf("Success: users sorted\n");

} else {

printf("Wrong option\n");

}

} else {

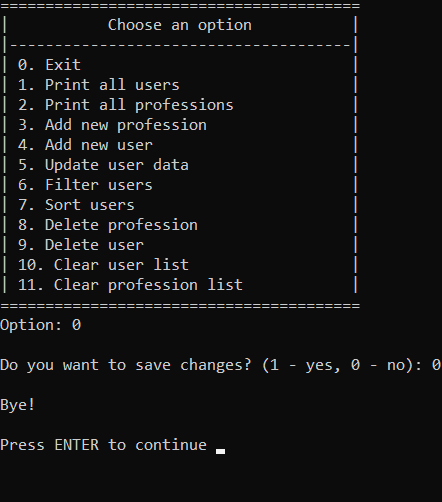
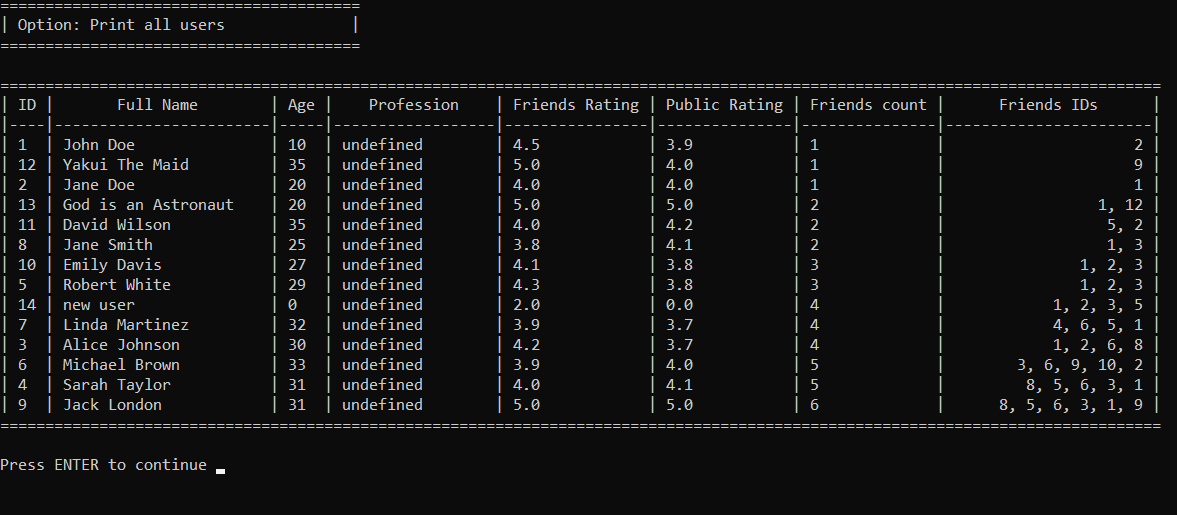
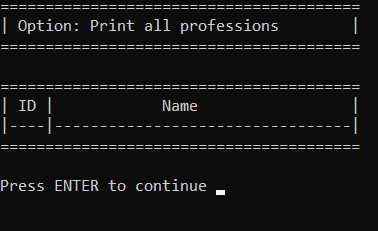
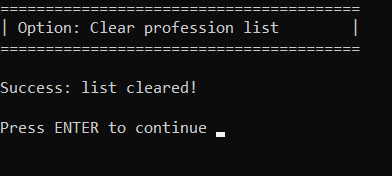
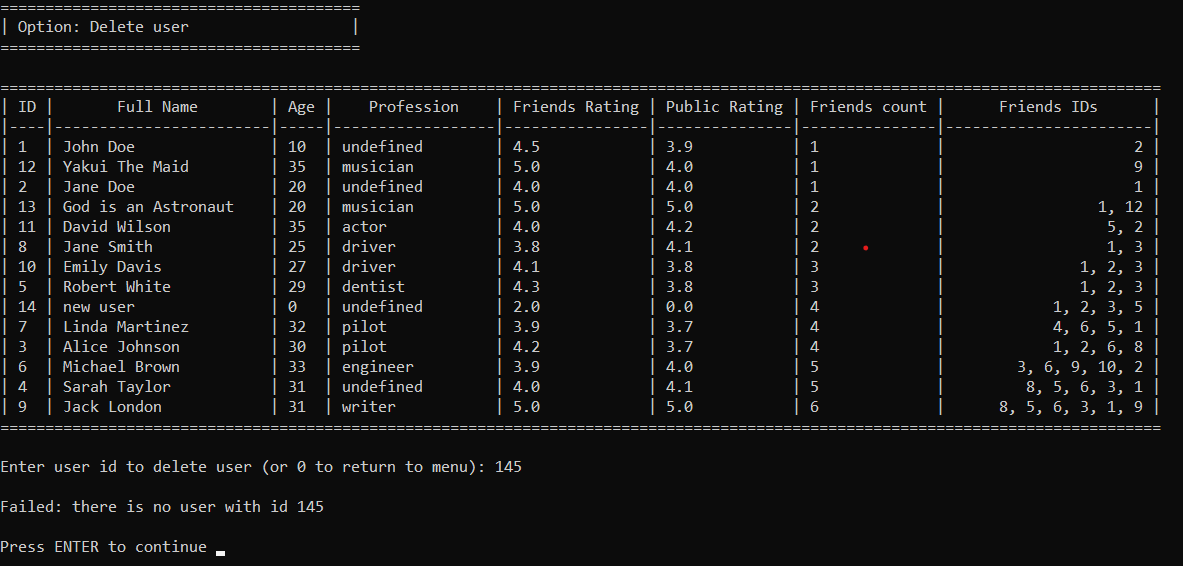
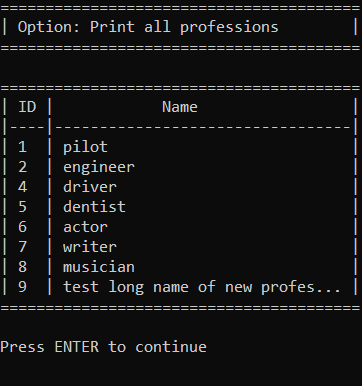
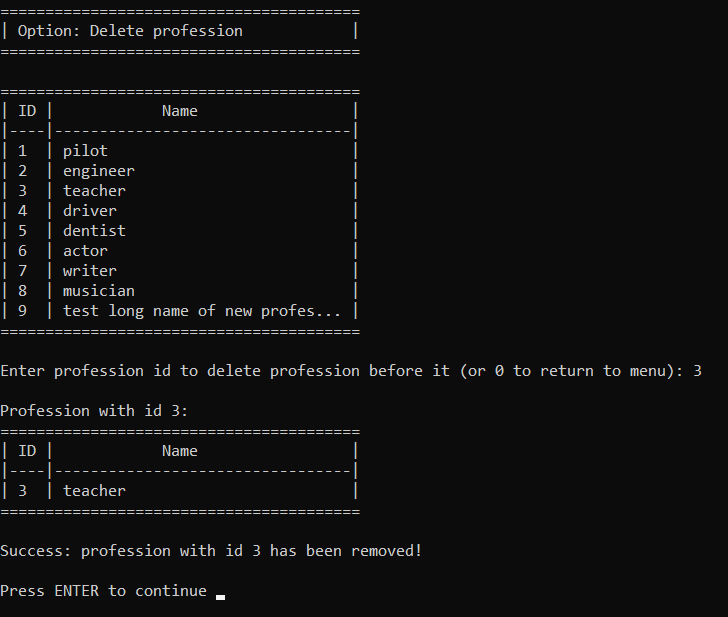
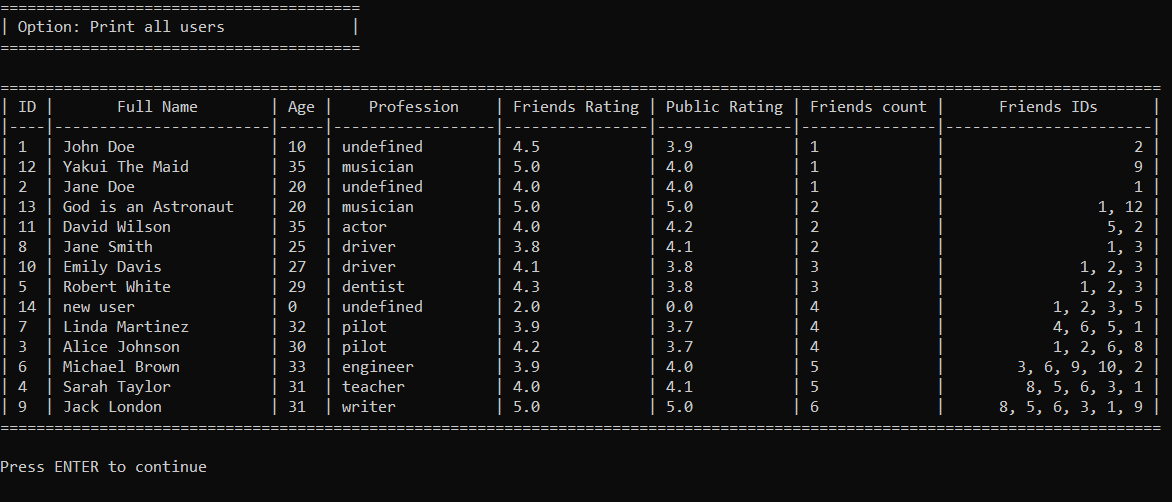
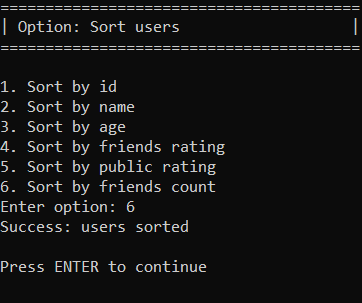
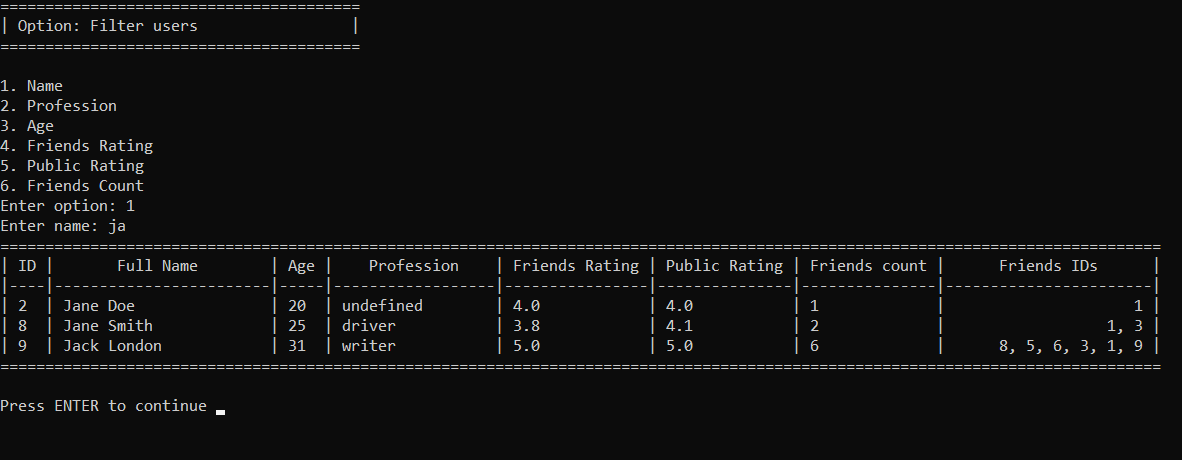
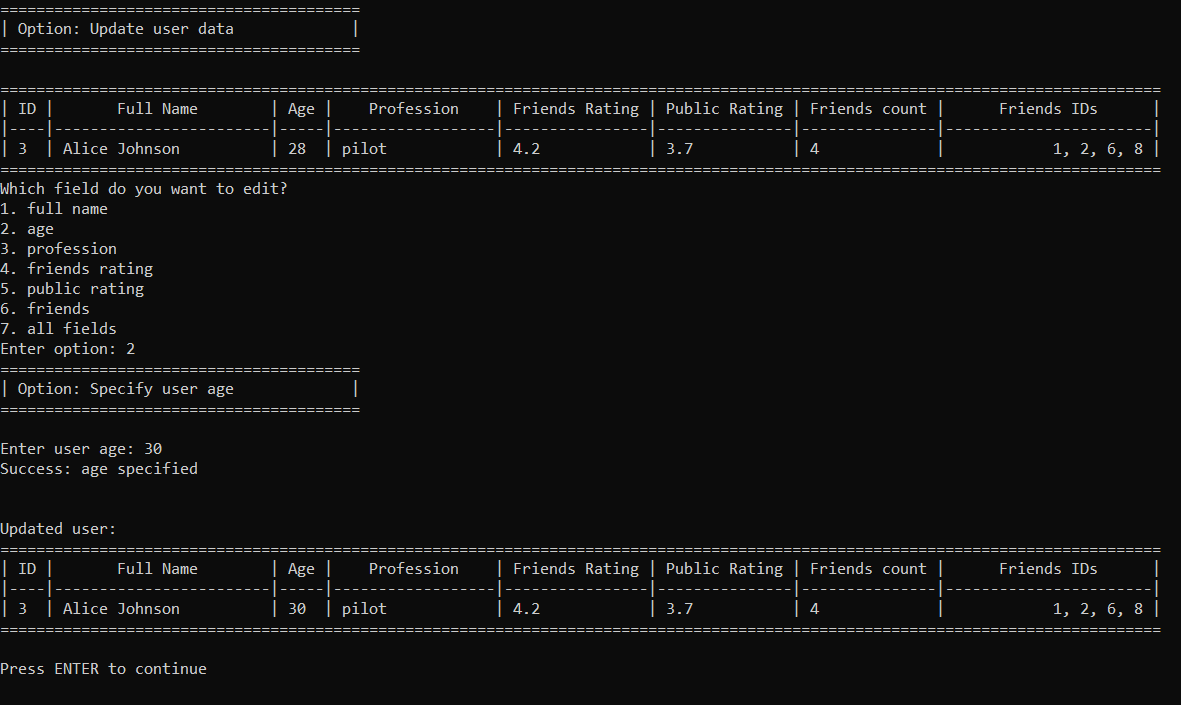
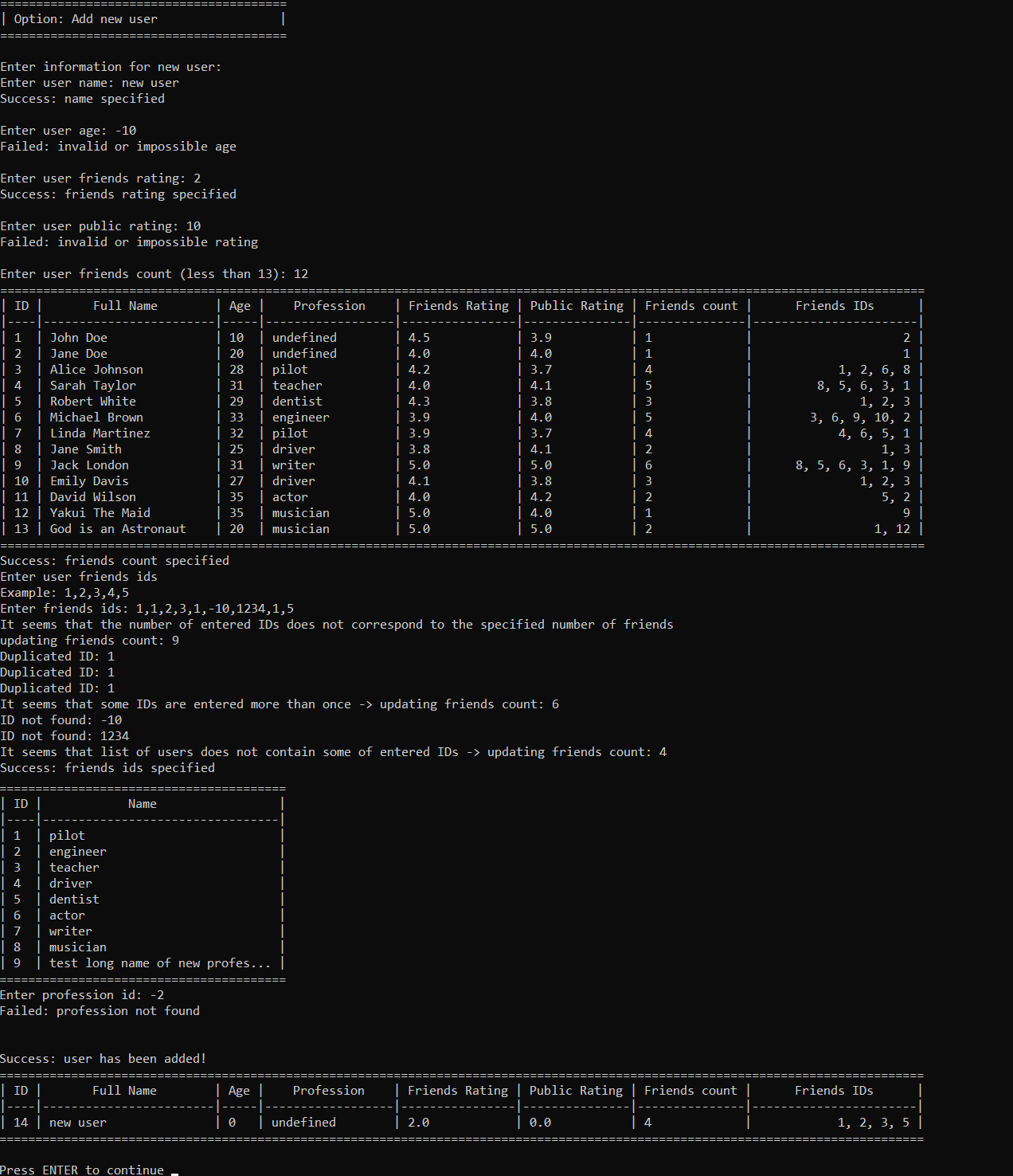
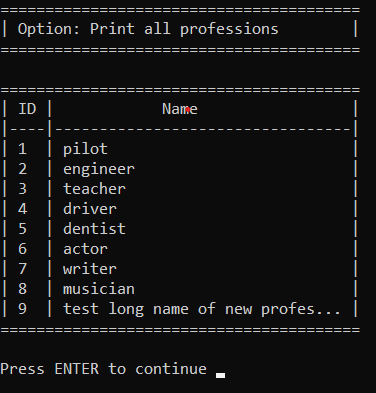
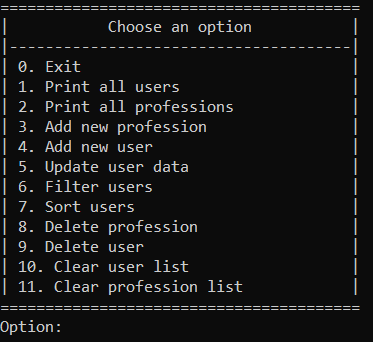
printf("The list of users is empty\n");

printf("You can add new user in menu with option 5\n");

}

}

**Примеры выполнения программы:**



**Заключение:**

Заголовочные файлы:

Заголовочный файл <stdio.h>

* printf
* fgets
* scanf
* sprintf
* perror
* fprintf
* fopen
* fclose
* rewind
* system

Заголовочный файл <stdlib.h>

* malloc
* free
* atoi
* atof
* qsort
* rand
* srand

Заголовочный файл <string.h>

* strcpy
* strncpy
* strcat
* strcmp
* strlen
* strtok

Заголовочный файл <time.h>

* time
* localtime
* strftime
* time\_t
* tm

Заголовочный файл <ctype.h>

* tolower

**Выводы:**

В результате выполнения работы была изучана работа со структурами в языке С и получены практические навыки в создании электронных картотек.