**Исходный текст программы**

#include<stdio.h>

#include<stdlib.h>

#include<string.h>

#include<ctype.h>

#define MAXLEN 256

typedef struct userStruct {

int id;

char\* fullName;

int age;

char\* profession;

float friendsRating;

float publicRating;

int friendsCount;

int friendsId[MAXLEN];

struct userStruct\* next;

} User;

typedef struct LHead {

int last\_id;

int isFriendsSorted;

struct userStruct\* first;

struct userStruct\* last;

} Head;

/\*\* for work with linked list \*/

Head\* makeHead();

User\* makeNode(char\*\* str);

void addNode(Head\* my\_head, User\* new\_node);

User\* selectId(Head\* my\_head, int id);

void deleteNode(Head\* my\_head, User\* current\_node);

void deleteById(Head\* head);

void addUser(Head\* head);

void freeStruct(User\* user);

void freeList(Head\* my\_head);

/\*\* utils \*/

void bubbleSortByField(Head \*my\_head, char\* desc);

void swapNodes(User\*\* prevNode, User\* a, User\* b);

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

void filterList(Head\* head);

void clearList(Head\* head);

char\*\* simpleSplit(char\* str, int length);

void simpleSplitInt(User\* user, const char\* str, int isManual, int idList[], int usersCount);

void trim(char\* str);

/\*\* output \*/

void printHeader();

void printUser(User\* user);

void printAllUsers(Head\* my\_head);

void pressEnterToContinue();

void clearConsole();

int main() {

Head\* head = NULL;

User\* user = NULL;

int slen, i, n, count, option;

char temp[MAXLEN];

char\*\* splitArray;

FILE\* file;

head = makeHead();

n = 0;

count = 0;

file = fopen("index.csv", "r");

if (file != NULL) {

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

rewind(file);

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

fgets(temp, MAXLEN, file);

slen = strlen(temp);

temp[slen - 1] = '\0';

splitArray = simpleSplit(temp, slen);

if (splitArray != NULL) {

user = makeNode(splitArray);

if (user != NULL) {

addNode(head, user);

} else {

puts("Structure not allocated!");

i = n;

}

} else {

puts("Error data reading!");

i = n;

}

}

fclose(file);

} else {

perror("Failed to open file");

printf("\n");

}

if (count == n) {

bubbleSortByField(head, "id");

clearConsole();

do {

printf("Choose option and press ENTER:\n"

"1. Print all users\n"

"2. Sort users by friends count\n"

"3. Sort users by id\n"

"4. Add new user\n"

"5. Filter users by name or profession\n"

"6. Delete user before specified id\n"

"7. Clear list\n"

"8. Exit\n\n");

printf("Option: ");

scanf("%d", &option);

getchar();

switch (option) {

case 1:

clearConsole();

printAllUsers(head);

pressEnterToContinue();

break;

case 2:

clearConsole();

bubbleSortByField(head, "friends");

printf("Sorted!\n");

head->isFriendsSorted = 1;

pressEnterToContinue();

break;

case 3:

clearConsole();

bubbleSortByField(head, "id");

printf("Sorted!\n");

head->isFriendsSorted = 0;

pressEnterToContinue();

break;

case 4:

clearConsole();

addUser(head);

pressEnterToContinue();

break;

case 5:

clearConsole();

filterList(head);

pressEnterToContinue();

break;

case 6:

clearConsole();

if (head->first != NULL) {

deleteById(head);

} else {

printf("The list is cleared and there is no users to remove!\n"

"You can add new users with command 4 in menu!\n");

}

pressEnterToContinue();

break;

case 7:

clearConsole();

if (head->first != NULL) {

clearList(head);

printf("The list is now cleared!\n");

} else {

printf("The list is already empty!\n");

}

pressEnterToContinue();

case 8:

clearConsole();

break;

default:

clearConsole();

break;

}

} while (option != 8);

} else {

puts("An error occurred while running the program!");

}

freeList(head);

return 0;

}

Head\* makeHead() {

Head\* ph = NULL;

ph = (Head\*)malloc(sizeof(Head));

if (ph != NULL) {

ph->last\_id = 0;

ph->first = NULL;

ph->last = NULL;

ph->isFriendsSorted = 0;

}

return ph;

}

User\* makeNode(char\*\* str) {

int mock[] = {0};

User \*user = NULL;

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

if (user != NULL) {

user->id = atoi(str[0]);

free(str[0]);

user->fullName = str[1];

user->age = atoi(str[2]);

free(str[2]);

user->profession = str[3];

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

free(str[4]);

user->publicRating = atof(str[5]);

free(str[5]);

user->friendsCount = atoi(str[6]);

free(str[6]);

simpleSplitInt(user, str[7], 0, mock, 0);

free(str[7]);

free(str);

user->next = NULL;

}

return user;

}

void addNode(Head\* my\_head, User\* new\_node) {

if (my\_head->first == NULL) {

my\_head->last\_id = 1;

my\_head->first = new\_node;

my\_head->last = new\_node;

} else {

my\_head->last\_id++;

new\_node->id = my\_head->last\_id;

new\_node->next = my\_head->first;

my\_head->first = new\_node;

}

}

User \*selectId(Head \*my\_head, int id) {

User \*q = NULL;

int maxId, isFound;

isFound = 0;

q = my\_head->first;

maxId = my\_head->last\_id;

if ((id > 0) && (id <= maxId) && q != NULL) {

do {

if (q->id == id) {

isFound = 1;

} else {

q = q->next;

}

} while (q != NULL && !isFound);

}

if (!isFound) {

q = NULL;

}

return q;

}

void deleteNode(Head \*my\_head, User \*current\_node) {

User \*q;

int isFound = 0;

q = my\_head->first;

if (current\_node == q) {

my\_head->first = current\_node->next;

current\_node->next = NULL;

freeStruct(current\_node);

} else {

while (q != NULL && !isFound) {

if (q->next == current\_node) {

if (current\_node == my\_head->last) {

my\_head->last = q;

}

q->next = current\_node->next;

current\_node->next = NULL;

freeStruct(current\_node);

isFound = 1;

} else {

q = q->next;

}

}

}

if (my\_head->first != NULL) {

my\_head->last\_id = my\_head->last->id;

} else {

my\_head->last\_id = 0;

}

}

void deleteById(Head\* head) {

User\* q;

int id, i, idFlag;

bubbleSortByField(head, "id");

q = head->first;

printf("Enter the users id to remove the user BEFORE it\n"

"(enter 0 if you don't want to do it)\n\n"

"Here is list of ids:\n[");

while (q != NULL) {

printf("%d", q->id);

if (q->next != NULL) {

printf(", ");

}

q = q->next;

}

printf("]\n\n");

printf("Example:\n"

"id: 2 -> user with id 1 will be removed\n\n"

"id: ");

scanf("%d", &id);

getchar();

if (id != 0) {

q = selectId(head, id - 1);

if (q != NULL) {

deleteNode(head, q);

printf("\nSuccess: user with id %d has been removed!\n", id - 1);

q = head->first;

while (q != NULL) {

idFlag = 0;

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

if (q->friendsId[i] == (id - 1)) {

idFlag = 1;

q->friendsCount--;

q->friendsId[q->friendsCount] = 0;

}

if (idFlag) {

q->friendsId[i] = q->friendsId[i + 1];

}

}

q = q->next;

}

} else {

printf("Failed: there is no user with id %d.\n", id - 1);

}

}

if (head->isFriendsSorted) {

bubbleSortByField(head, "friends");

}

}

void addUser(Head \*head) {

User \*newUser, \*q;

char tempStr[MAXLEN];

int idList[MAXLEN] = {0};

int tempAge, i;

int usersCount = 0;

float tempFriendsRating, tempPublicRating;

int tempFriendsCount;

newUser = malloc(sizeof(User));

if (newUser == NULL) {

perror("Memory allocation failed");

} else {

newUser->next = NULL;

newUser->id = head->last\_id + 1;

printf("Enter full name: ");

newUser->fullName = malloc(MAXLEN \* sizeof(char));

if (newUser->fullName == NULL || fgets(newUser->fullName, MAXLEN, stdin) == NULL) {

perror("Failed to read full name or allocate memory");

free(newUser);

} else {

trim(newUser->fullName);

printf("\nEnter age: ");

scanf("%d", &tempAge);

getchar();

newUser->age = tempAge > 0 && tempAge < 256 ? tempAge : 0;

if (tempAge >= 256) puts("No way human live more than 256 years -> setting 0 automatically");

printf("\nEnter profession: ");

newUser->profession = malloc(MAXLEN \* sizeof(char));

if (newUser->profession == NULL || fgets(newUser->profession, MAXLEN, stdin) == NULL) {

perror("Failed to read image filename or allocate memory");

free(newUser->fullName);

free(newUser);

} else {

trim(newUser->profession);

printf("\nEnter friends rating (float number less than 5): ");

if (scanf("%f", &tempFriendsRating) == 1 && tempFriendsRating <= 5) {

newUser->friendsRating = tempFriendsRating;

} else {

puts("Invalid friends rating -> setting 0 automatically");

newUser->friendsRating = 0;

}

getchar();

printf("\nEnter public rating (float number less than 5): ");

if (scanf("%f", &tempPublicRating) == 1 && tempPublicRating <= 5) {

newUser->publicRating = tempPublicRating;

} else {

puts("Invalid public rating -> setting 0 automatically");

newUser->publicRating = 0;

}

getchar();

if (head->first != NULL) {

usersCount = 0;

bubbleSortByField(head, "id");

q = head->first;

while (q != NULL) {

idList[usersCount++] = q->id;

q = q->next;

}

printf("\nEnter friends count (less than %d): ", usersCount);

if (scanf("%d", &tempFriendsCount) == 1 && tempFriendsCount <= MAXLEN && tempFriendsCount >= 0 && tempFriendsCount <= usersCount) {

newUser->friendsCount = tempFriendsCount;

} else {

puts("Invalid friends count -> setting 0 automatically");

tempFriendsCount = 0;

newUser->friendsCount = 0;

}

getchar();

} else {

newUser->friendsCount = 0;

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

newUser->friendsId[i] = 0;

}

}

if (tempFriendsCount > 0) {

printf("\nEnter friends IDs (example: 1,2,3,4)\n");

printf("Available IDs: [%d", idList[0]);

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

printf(",%d", idList[i]);

}

printf("]\n\n");

printf("IDs: ");

scanf("%s", tempStr);

getchar();

simpleSplitInt(newUser, tempStr, 1, idList, usersCount);

}

addNode(head, newUser);

if (head->isFriendsSorted) {

bubbleSortByField(head, "friends");

} else {

bubbleSortByField(head, "id");

}

printf("\nNew user successfully added!\n");

}

}

}

}

void freeStruct(User \*user) {

if (user != NULL) {

free(user->fullName);

user->fullName = NULL;

free(user->profession);

user->profession = NULL;

free(user);

}

}

void freeList(Head\* my\_head) {

User \*q, \*q1;

/\* there are two pointers here because we need to remember

the next value of the structure we are going to free \*/

q = my\_head->first;

while (q != NULL) {

q1 = q->next;

freeStruct(q);

q = q1;

}

free(my\_head);

}

/\*\*

\* Sorts a linked list of users either by their ID or by their friend count in ascending order.

\* This function implements a bubble sort algorithm that iterates through the list, comparing

\* adjacent nodes based on the specified field (`desc`). If `desc` is "id", nodes are sorted

\* by their ID in ascending order else nodes are sorted by their friendsCount in descending order.

\*

\* The use of a double pointer (\*\*ptr1) for node swapping is crucial here. It allows directly

\* modifying the pointer to the current node in the list (either the `first` field of the head

\* or the `next` field of a node). This direct manipulation eliminates the need for a separate

\* case to update the head of the list when the first two nodes are swapped and simplifies

\* swapping nodes in general by adjusting the pointer that points to the current node rather

\* than the node itself.

\*

\* @param my\_head A pointer to the head of the list, which contains pointers to the first

\* and last nodes of the list.

\* @param desc A string that specifies the field by which the list should be sorted:

\* "id" for sorting by ID in ascending order, else for sorting by

\* friendsCount in descending order.

\*/

void bubbleSortByField(Head \*my\_head, char\* desc) {

int swapped;

User \*\*ptr1;

User \*lptr = NULL;

if (my\_head->first != NULL) {

do {

swapped = 0;

ptr1 = &(my\_head->first);

while ((\*ptr1)->next != lptr) {

/\* if current element is greater than next -> we swap them \*/

if ((strcmp(desc, "id") == 0) ?

(\*ptr1)->id > (\*ptr1)->next->id :

(\*ptr1)->friendsCount < (\*ptr1)->next->friendsCount) {

swapNodes(ptr1, \*ptr1, (\*ptr1)->next);

swapped = 1;

}

/\* getting next element (if current was greater -> we will get it again) \*/

ptr1 = &((\*ptr1)->next);

}

lptr = \*ptr1;

} while (swapped);

lptr = my\_head->first;

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

lptr = lptr->next;

}

my\_head->last = lptr;

}

}

/\*\*

\* Swaps two nodes in a linked list.

\* This function updates the previous node's next pointer to point to the second node, effectively

\* swapping the two nodes in the list. It makes use of a double pointer to the previous node's next

\* field (\*\*prevNode) to directly modify the link between nodes, facilitating the swap operation

\* without additional steps to handle special cases, such as swapping the head of the list.

\*

\* @param prevNode A double pointer to the previous node's next field, pointing to the first of

\* the two nodes to be swapped.

\* @param a A pointer to the first node to be swapped.

\* @param b A pointer to the second node to be swapped, immediately following the first.

\*/

void swapNodes(User \*\*prevNode, User \*a, User \*b) {

(\*prevNode) = b;

a->next = b->next;

b->next = a;

}

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 filterList(Head\* head) {

User\* q = NULL;

User\* q1 = NULL;

char ask;

int j;

char temp[MAXLEN];

printf("You can sort users by either name or profession. Choose one option (1 or 2): ");

scanf("%c", &ask);

getchar();

if (ask != '1' && ask != '2') {

printf("invalid option");

} else {

clearConsole();

if (ask == '1') {

printf("Enter the user name: ");

} else {

printf("Enter the profession: ");

}

scanf("%s", temp);

getchar();

printf("\n");

printHeader();

j = 0;

q = head->first;

while (q != NULL) {

if (startsWithIgnoreCase((ask == '2') ? q->profession : q->fullName, temp)) {

printUser(q);

j++;

}

q1 = q->next;

q = q1;

}

if (j == 0) {

printf("\nNo user seems to match your input.\n");

}

}

}

void clearList(Head\* head) {

User\* q;

User\* q1;

q = head->first;

while (q != NULL) {

q1 = q->next;

deleteNode(head, q);

q = q1;

}

}

char \*\*simpleSplit(char \*str, int length) {

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] == ';') count++;

}

count++;

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

if (result == NULL) {

perror("Memory allocation failed");

} 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 simpleSplitInt(User\* user, const char \*str, int isManual, int idList[], int usersCount) {

int count = 0, cnt = 0;

int start = 0;

int i, len, flag, n, j;

char tempStr[MAXLEN];

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

if (str[i] == ',') cnt++;

}

cnt++;

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

user->friendsId[i] = 0;

}

flag = 1;

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

flag = isManual ? 0 : 1;

for (j = 0; j < usersCount && !flag; j++) {

if (n == idList[j]) {

flag = 1;

}

}

user->friendsId[count++] = n;

start = i + 1;

}

}

if (!flag) {

puts("It seems that among friends there is a user ID that is not in the database -> setting 0 friends automatically");

user->friendsCount = 0;

}

if (count < user->friendsCount) {

printf("It seems that the number of entered IDs does not correspond to the specified number of friends -> updating friends count: %d\n", count);

user->friendsCount = count;

}

}

void trim(char \*str) {

int i = 0;

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

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

str[i] = '\0';

i = MAXLEN;

}

}

}

void printHeader() {

printf("%-3s %-20s %-5s %-15s %-15s %-15s %-15s %-20s\n",

"ID", "Full Name", "Age", "Profession", "Friends Rating", "Public Rating", "Friends Count", "Friends IDs");

}

void printUser(User \*user) {

int i;

printf("%-3d %-20s %-5d %-15s %-15.1f %-15.1f %-15d ",

user->id, user->fullName, user->age, user->profession, user->friendsRating, user->publicRating, user->friendsCount);

printf("[");

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

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

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

printf(", ");

}

}

printf("]\n");

}

void printAllUsers(Head\* my\_head) {

User \*q;

printHeader();

q = my\_head->first;

while (q != NULL) {

printUser(q);

q = q->next;

}

}

void pressEnterToContinue() {

printf("\nPress ENTER to continue ");

getchar();

clearConsole();

}

void clearConsole() {

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

system("cls");

#else

system("clear");

#endif

}

**Контрольные примеры**

**Пример:**

Choose option and press ENTER:

1. Print all users

2. Sort users by friends count

3. Sort users by id

4. Add new user

5. Filter users by name or profession

6. Delete user before specified id

7. Clear list

8. Exit

Option: 1

ID Full Name Age Profession Friends Rating Public Rating Friends Count Friends IDs

1 John Doe 30 teacher 4.5 3.9 3 [2, 5, 7]

2 Jane Smith 25 engineer 3.8 4.1 2 [1, 3]

3 Alice Johnson 28 driver 4.2 3.7 4 [1, 2, 6, 8]

4 Michael Brown 33 pilot 3.9 4.0 5 [3, 6, 9, 10, 2]

5 Emily Davis 27 dentist 4.1 3.8 3 [1, 2, 3]

6 David Wilson 35 actor 4.0 4.2 2 [5, 2]

7 Linda Martinez 32 actor 3.9 3.7 4 [4, 6, 5, 1]

8 Robert White 29 teacher 4.3 3.8 3 [1, 2, 3]

9 Sarah Taylor 31 teacher 4.0 4.1 5 [8, 5, 6, 3, 1]

10 James Anderson 34 pilot 4.2 3.9 2 [1, 2]

11 Davidios Morgan 20 teacher 2.0 1.0 0 []

12 Casey Taylor 28 pilot 3.7 1.7 5 [6, 8, 13, 10, 15]

13 Jamie Jones 27 dentist 4.4 1.3 5 [8, 3, 11, 1, 7]

14 Charlie Williams 26 engineer 3.4 3.5 0 []

15 Sam Jones 26 driver 2.8 4.4 3 [3, 5, 2]

16 Jordan Miller 35 engineer 3.0 2.0 4 [1, 13, 4, 14]

17 Chris Moore 20 pilot 2.7 4.4 0 []

18 Charlie Williams 32 actor 3.1 2.9 1 [14]

19 Casey Wilson 20 dentist 3.1 2.7 0 []

20 Taylor Johnson 32 driver 4.1 2.8 4 [12, 7, 20, 10]

21 Cameron Moore 28 pilot 2.4 2.1 2 [7, 10]

22 Casey Moore 35 actor 3.9 1.8 2 [13, 15]

23 Charlie Moore 24 teacher 3.4 1.7 5 [10, 3, 25, 18, 22]

24 Chris Wilson 27 teacher 4.4 3.6 3 [7, 14, 20]

25 Cameron Wilson 32 dentist 4.3 2.6 5 [8, 14, 15, 23, 29]

26 Chris Johnson 21 pilot 2.5 1.5 0 []

27 Jamie Moore 27 dentist 2.9 4.4 5 [25, 1, 11, 31, 27]

28 Jamie Moore 24 driver 2.2 1.1 0 []

29 Jordan Jones 31 engineer 3.9 3.7 2 [27, 16]

30 Sam Wilson 26 driver 2.3 4.3 0 []

Press ENTER to continue

Choose option and press ENTER:

1. Print all users

2. Sort users by friends count

3. Sort users by id

4. Add new user

5. Filter users by name or profession

6. Delete user before specified id

7. Clear list

8. Exit

Option: 2

Sorted!

Press ENTER to continue

Choose option and press ENTER:

1. Print all users

2. Sort users by friends count

3. Sort users by id

4. Add new user

5. Filter users by name or profession

6. Delete user before specified id

7. Clear list

8. Exit

Option: 4

Enter full name: new User

Enter age: 12

Enter profession: pilot

Enter friends rating (float number less than 5): 4.23

Enter public rating (float number less than 5): 4.1

Enter friends count (less than 30): 12

Enter friends IDs (example: 1,2,3,4)

Available IDs: [4,9,12,13,23,25,27,3,7,16,20,1,5,8,15,24,2,6,10,21,22,29,18,11,14,17,19,26,28,30]

IDs: 1,2,3

It seems that the number of entered IDs does not correspond to the specified number of friends -> updating friends count: 3

New user successfully added!

Press ENTER to continue

Choose option and press ENTER:

1. Print all users

2. Sort users by friends count

3. Sort users by id

4. Add new user

5. Filter users by name or profession

6. Delete user before specified id

7. Clear list

8. Exit

Option: 6

Enter the users id to remove the user BEFORE it

(enter 0 if you don't want to do it)

Here is list of ids:

[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30]

Example:

id: 2 -> user with id 1 will be removed

id: 2

Success: user with id 1 has been removed!

Press ENTER to continue

Choose option and press ENTER:

1. Print all users

2. Sort users by friends count

3. Sort users by id

4. Add new user

5. Filter users by name or profession

6. Delete user before specified id

7. Clear list

8. Exit

Option: 6

Enter the users id to remove the user BEFORE it

(enter 0 if you don't want to do it)

Here is list of ids:

[2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30]

Example:

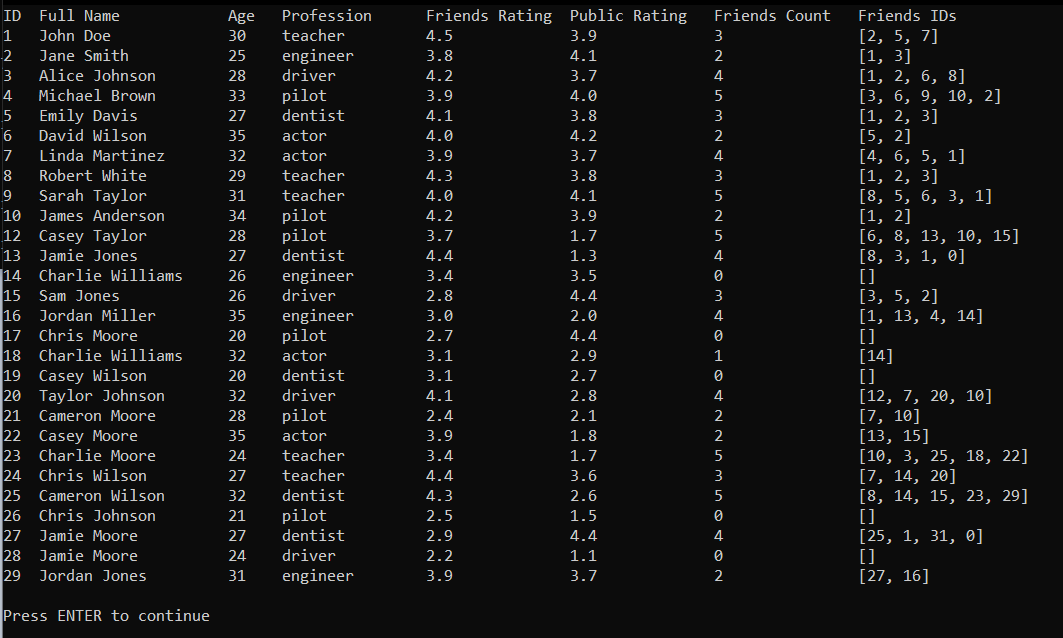
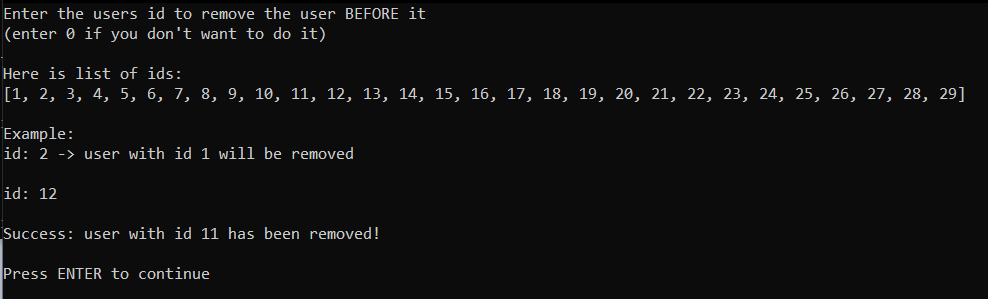
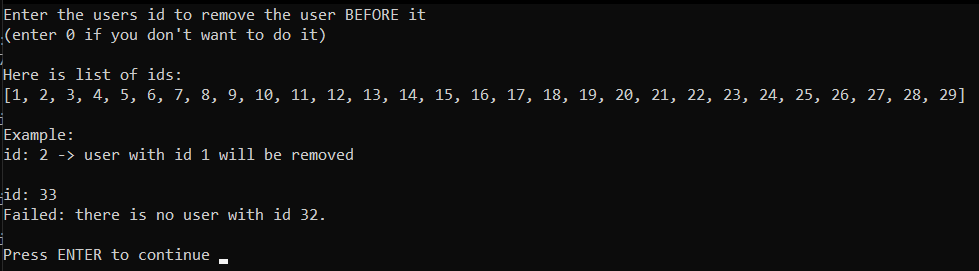
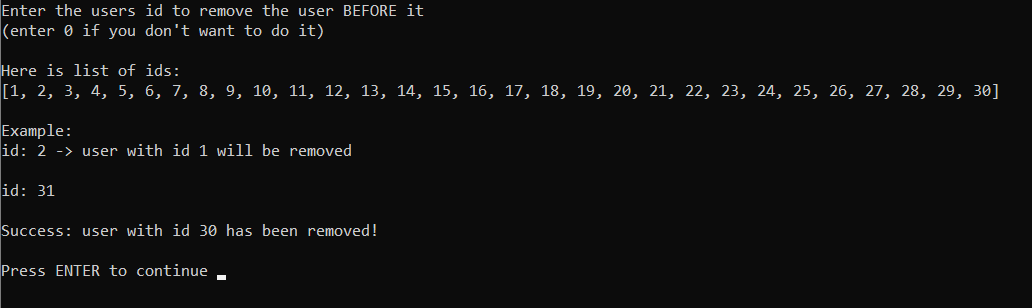
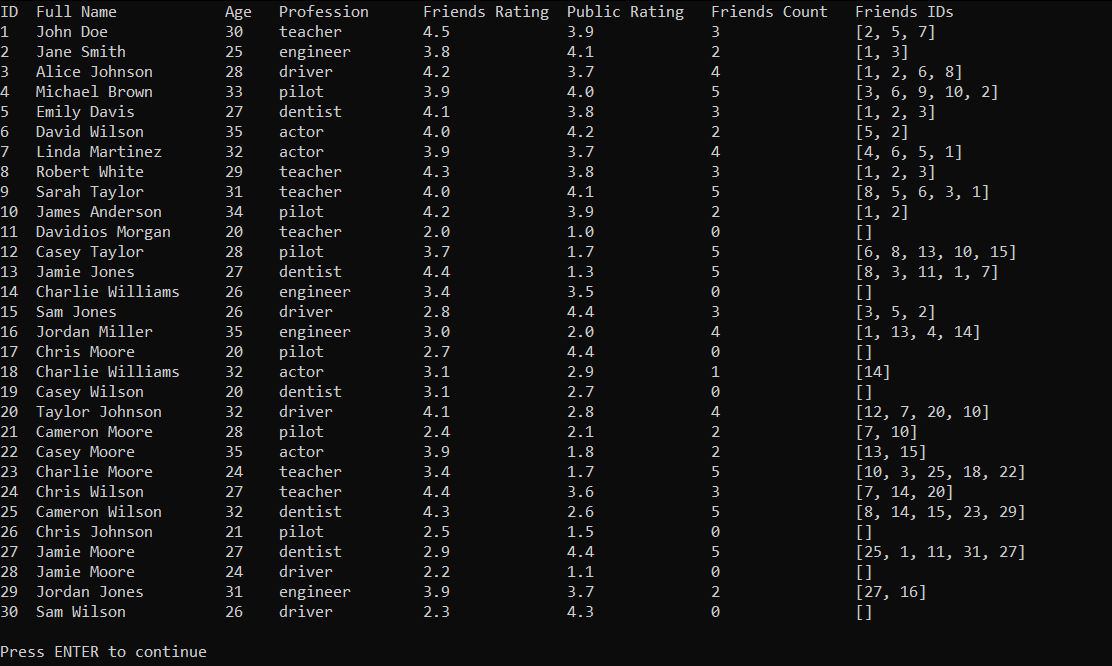
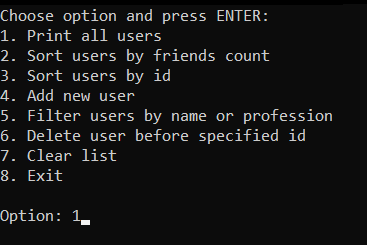
id: 2 -> user with id 1 will be removed

id: 2

Failed: there is no user with id 1.

Press ENTER to continue

**Примеры выполнения программы**

****

**Выводы.**

В ходе выполнения работы были получены практические навыки в работе с линейными односвязными списками в языке С.