**Исходный текст программы**

#include <stdio.h>

#include <stdlib.h>

#include <string.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 appOption(ProfessionHead\* professionHead, UserHead\* userHead, int option);

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

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

UserHead\* makeUserHead();

User\* makeUserNode(ProfessionHead\* pHead, UserHead\* uHead, 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);

void nullString(char str[MAXLEN]);

void trim(char str[MAXLEN]);

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

void inputIntrray(UserHead\* uHead, User\* user, char \*str, char sep);

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

void clearStdin();

int main() {

UserHead\* userHead = NULL;

ProfessionHead\* professionHead = NULL;

userHead = makeUserHead();

professionHead = makeProfessionHead();

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

appGUI(professionHead, userHead);

} else {

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

}

return 0;

}

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

int option;

readProfessions("professions.csv", professionHead);

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

do {

clearConsole();

printMenu();

scanf("%d", &option);

clearStdin();

if (option != 0) {

appOption(professionHead, userHead, option);

} else {

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("Delete profession before id");

deleteProfessionGUI(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) {

id--;

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");

}

}

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. Delete profession before id |\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));

}

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");

}

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) {

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");

}

if (count != n) {

perror("Failed to read from file");

freeProfessionList(head);

}

}

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

Profession\* q = NULL;

q = head->first;

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

q = q->next;

}

return q;

}

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");

}

return head;

}

User\* makeUserNode(ProfessionHead\* pHead, UserHead\* uHead, char\*\* str) {

User\* user = NULL;

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

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;

inputIntrray(uHead, user, str[6], ',');

} else {

user->friendsId = NULL;

}

free(str[6]);

free(str);

user->next = NULL;

user->prev = NULL;

} else {

perror("Memory allocation failed");

}

return 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 != NULL) {

free(user->fullName);

user->fullName = NULL;

if (user->friendsId != NULL) {

free(user->friendsId);

user->friendsId = NULL;

}

free(user);

}

}

void freeUserList(UserHead\* 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 = head->first;

while (q != NULL) {

q1 = q->next;

freeUserStruct(q);

q = q1;

}

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) {

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(pHead, head, splitArray);

if (user != NULL) {

pushBackUserNode(head, user);

}

}

}

fclose(file);

} else {

perror("Failed to open file");

}

if (count != n) {

perror("Failed to read from file");

freeUserList(head);

}

}

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");

} 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 inputIntrray(UserHead\* uHead, User\* user, char \*str, char sep) {

int enteredIdCount = 0, sepCount = 0;

int start = 0;

int i, len, isInputValid, n;

char tempStr[MAXLEN];

int enteredIds[MAXLEN];

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");

sepCount = 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;

}

}

}

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

if (user->friendsId == NULL) {

perror("Memory allocation failed");

} else {

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

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

}

}

}

void clearStdin() {

int c;

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

}

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;

}

**Контрольные примеры**

**Пример 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 | 3 | 2, 5, 7 |

| 2 | Jane Doe | 20 | undefined | 4.0 | 4.0 | 10 | 1, 2, 3, 4, 5, 6, ... |

| 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 |

=================================================================================================================================

========================================

| Option: Print all professions |

========================================

========================================

| ID | Name |

|----|---------------------------------|

| 1 | pilot |

| 2 | engineer |

| 3 | teacher |

| 4 | driver |

| 5 | dentist |

| 6 | actor |

| 7 | writer |

| 8 | musician |

========================================

========================================

| Option: Delete profession before id |

========================================

========================================

| ID | Name |

|----|---------------------------------|

| 1 | pilot |

| 2 | engineer |

| 3 | teacher |

| 4 | driver |

| 5 | dentist |

| 6 | actor |

| 7 | writer |

| 8 | musician |

========================================

Enter profession id to delete profession before it (or 0 to return to menu): 1

Failed: there is no profession with id 0

========================================

| Option: Delete profession before id |

========================================

========================================

| ID | Name |

|----|---------------------------------|

| 1 | pilot |

| 2 | engineer |

| 3 | teacher |

| 4 | driver |

| 5 | dentist |

| 6 | actor |

| 7 | writer |

| 8 | musician |

========================================

Enter profession id to delete profession before it (or 0 to return to menu): 2

Profession with id 1:

========================================

| ID | Name |

|----|---------------------------------|

| 1 | pilot |

========================================

Success: profession with id 1 has been removed!

========================================

| 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 | 3 | 2, 5, 7 |

| 2 | Jane Doe | 20 | undefined | 4.0 | 4.0 | 10 | 1, 2, 3, 4, 5, 6, ... |

| 3 | Alice Johnson | 28 | undefined | 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 | undefined | 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 |

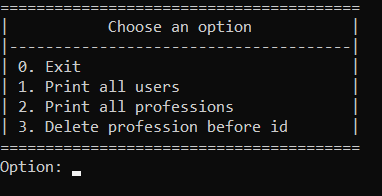
| 10 | Emily Davis | 27 | driver | 4.1 | 3.8 | 3 | 1, 2, 3 |

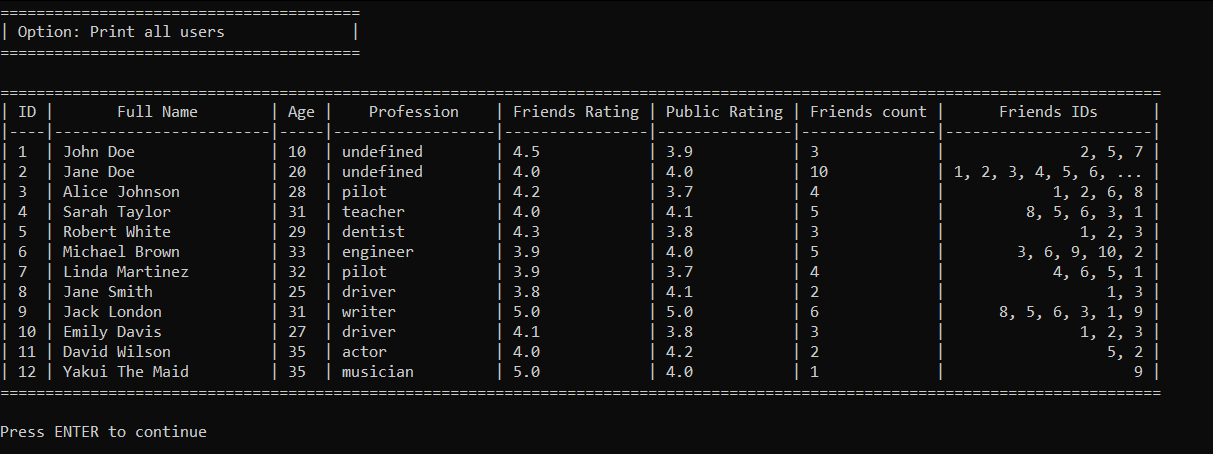
| 11 | David Wilson | 35 | actor | 4.0 | 4.2 | 2 | 5, 2 |

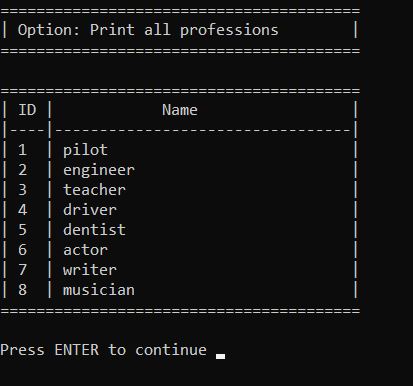
| 12 | Yakui The Maid | 35 | musician | 5.0 | 4.0 | 1 | 9 |

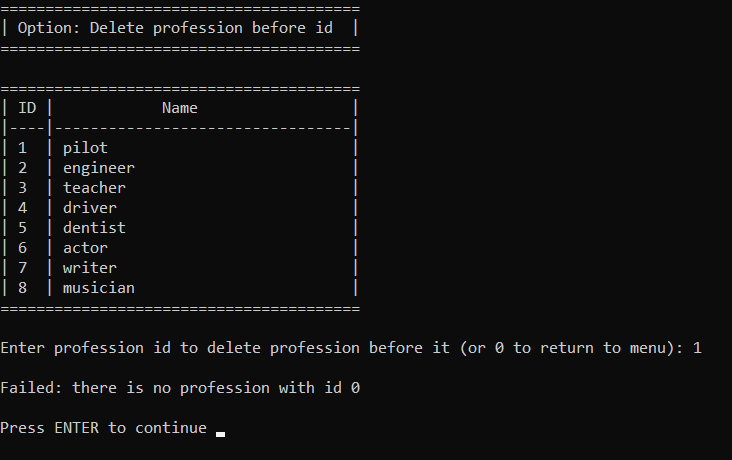
=================================================================================================================================

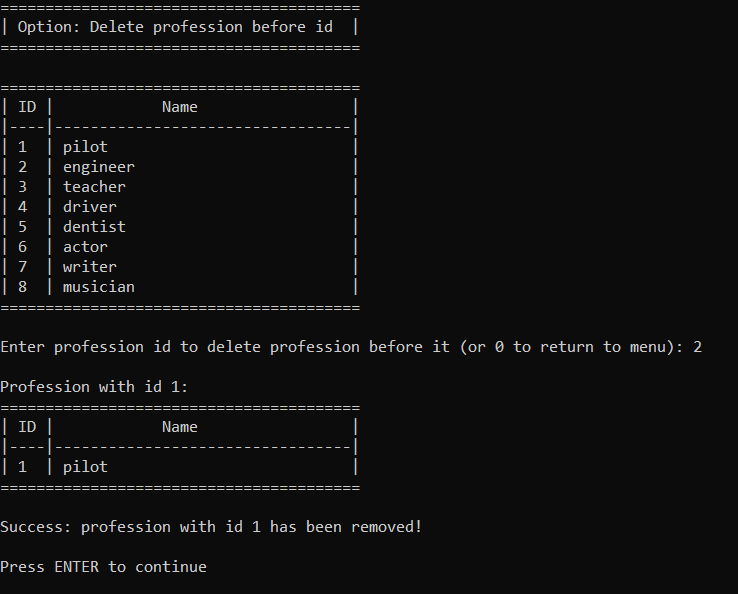
**Примеры выполнения программы**

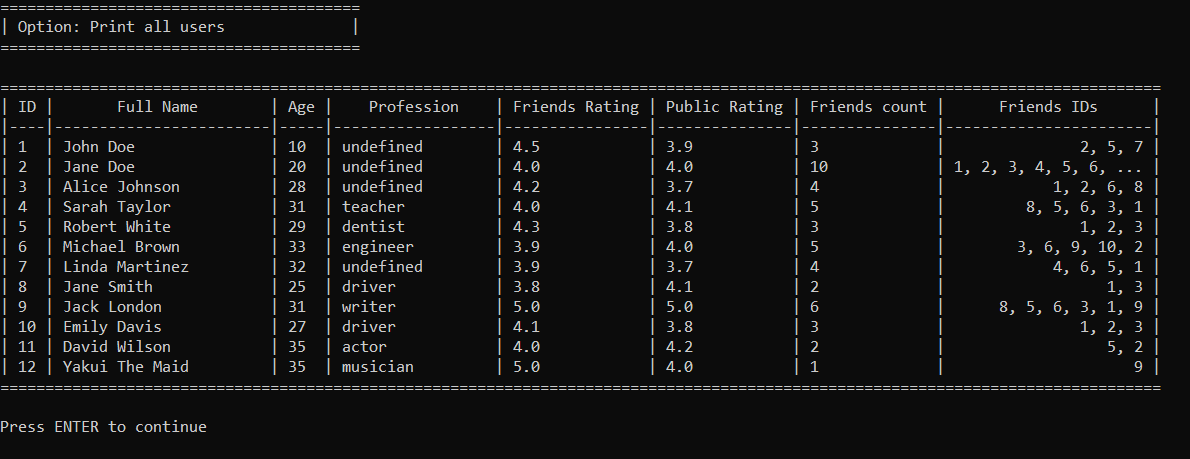
****

****

****

****

****

****

**Выводы.**

В результате выполнения работы были получены практические навыки работы с линейными двусвязными списками в языке С.