**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра МО ЭВМ**

Курсовая РАБОТА

**по дисциплине «Программирование»**

Тема: Двунаправленный список

|  |  |  |
| --- | --- | --- |
| Студент гр. 6304 |  | Васильев А.А. |
| Преподаватель |  | Берленко Т.А. |

Санкт-Петербург

2016

**ЗАДАНИЕ**

**на курсовую работу**

Студент Васильев А.А. Иванов И.И.

Группа 6304

Тема работы: Двунаправленный список

Содержание пояснительной записки:

* Анотация
* Содержание
* Введение (цель работы, постановка задачи, индивидуальное задание)
* Описание функций создания списка и работы с ним
* Работа с make и github
* Заключение
* Приложения

Предполагаемый объем пояснительной записки:

Не менее 20 страниц.

Дата выдачи задания: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Дата сдачи реферата: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Дата защиты реферата: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Студент \_\_\_\_\_\_\_\_\_\_\_\_\_\_ Васильев А.А.

Преподаватель \_\_\_\_\_\_\_\_\_\_\_\_\_\_ Берленко Т.А.

АННОТАЦИЯ

В данной курсовой работе подробно описывается процесс создания двунаправленного линейного списка и API для работы со списком (создание/удаление элемента, поиск элементов по значениям полей, добавление нового элемента в конец/начало списка). Для реализации данных функций, в работе используются широкие возможности языка программирования С (динамическая память, сложные типы данных, указатели и т.д.). Также описывается работа с утилитой для сборки проекта “make” и с репозиторием “github”.

Содержание

[АННОТАЦИЯ 3](#_Toc470127207)

[ВВЕДЕНИЕ 5](#_Toc470127208)

[Цель работы 5](#_Toc470127209)

[Формулировка задачи 5](#_Toc470127210)

[Индивидуальное задание 6](#_Toc470127211)

[ХОД РАБОТЫ 7](#_Toc470127212)

[Создание отдельных элементов проекта 7](#_Toc470127213)

[Работа с “make” и сборка проекта 15](#_Toc470127214)

[Работа с “github” 18](#_Toc470127215)

[ЗАКЛЮЧЕНИЕ 20](#_Toc470127216)

[ПРИЛОЖЕНИЕ 1 21](#_Toc470127217)

ВВЕДЕНИЕ

Цель работы

* Создание двунаправленного связанного списка, составными элементами которого являются структуры.
* Создание API для работы с созданным списком.
* Закрепление на практике полученных знаний о языке С.
* Закрепление знаний о работе с утилитой “make” и репозиторием “github”.

Формулировка задачи

Требуется создать двунаправленный список музыкальных композиций MusicalComposition и **api** ( ***a****pplication****p****rogramming****i****nterface - в данном случае набор функций*) для работы со списком.

Структура элемента списка (тип - MusicalComposition):

* name - строка , название композиции.
* author – строка, автор композиции/музыкальная группа.
* year - целое число, год создания.

Функция для создания элемента списка:

* MusicalComposition\* createMusicalComposition(char\* name, char\* author, int year).

Функции для работы со списком:

* MusicalComposition\* createMusicalCompositionList(char\*\* array\_names, char\*\* array\_authors, int\* array\_years, int n); // создает список музыкальных композиций MusicalCompositionList, в котором:
  + ***n****- длина массивов****array\_names****,****array\_authors****,****array\_years****.*
  + поле **name** первого элемента списка соответствует первому элементу списка array\_names (**array\_names[0]**).
  + поле **author** первого элемента списка соответствует первому элементу списка array\_authors (**array\_authors[0]**).
  + поле **year** первого элемента списка соответствует первому элементу списка array\_authors (**array\_years[0]**).
  + Функция возвращает указатель на первый элемент списка*.*
* void push(MusicalComposition\*  head); // добавляет в конец списка выбранноеколичествоэлементов.
* void removeEl (MusicalComposition\*  head, char\* name\_for\_remove); // удаляет элемент **element** списка, у которого значение **name** равно значению  **name\_for\_remove.**
* int count(MusicalComposition\*  head); //возвращает количество элементов списка.
* void print\_names(MusicalComposition\*  head); //Выводит названия композиций, автора и год создания.

Индивидуальное задание

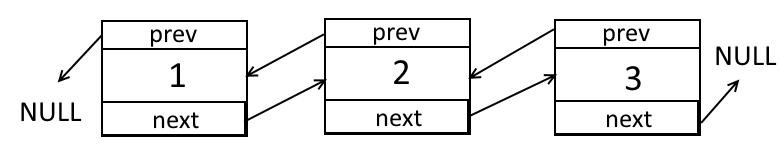
* Добавить в начало списка еще 3 элемента (имя, автор, год - произвольные)
* Удалить все элементы чьи года не превышают n.

ХОД РАБОТЫ

Создание отдельных элементов проекта

1. **struct.h**

Двунаправленный линейный список имеет следующую структуру:



Поэтому элементы списка должны содержать поля, указывающие на связанные структуры:

struct MusicalComposition{ //Объявляем структуру

char name[80];

char author[80];

int year;

struct MusicalComposition \*next, \* prev; //Указатели на

}; //связанные элементы.

typedef struct MusicalComposition MusicalComposition;

//Присваиваем, для удобства, новое имя типу-структуре

1. **get\_name**

Так как названия групп и композиций могут содержать несколько слов, то требуется функция, которая считывает символы до знака переноса строки. Функция **gets()** неявляется безопасной и компилятор **gcc** выдает предупреждение об этом. Поэтому создадим функцию, которая динамически выделяет память под каждый символ и возвращает указатель на первый.

char\* get\_name()

{

char\* ret = NULL;

char ch;

int k = 0;

/\*Пока не будет введен символ переноса строки память будет перераспределяться и в выделенную ячейку будет записываться введеный символ\*/

while((ch = getchar())!='\n')

{

ret = (char\*)realloc(ret, (k+2)\*sizeof(char));

ret[k++] = ch;

}

if (k==0) return " "; //Если сразу введен '\n', то возвращается пробел

ret[k--] = '\0';

while(ret[k] == ' ' || ret[k] == '\t'){ //Данная часть удаляет пробелы и

ret[k--] = '\0'; //табуляции в конце строки

}

return ret;

}

1. **createMusicalComposition**

Опишем функцию, которая создает структуру из переданных ей данных:

MusicalComposition\* createMusicalComposition(char\* name, char\* author, int year)

{

MusicalComposition\* tmp=NULL;

tmp =(MusicalComposition\*)malloc(sizeof(MusicalComposition));

strcpy(tmp->name, name);

strcpy(tmp->author, author);

tmp->year = year;

tmp->next = NULL;

tmp->prev = NULL;

return tmp;

}

Сначала выделяется память под структуру, затем функцией **strcpy** значения передаваемые в функцию копируются в соответствующие поля структуры. Указатели на связанные элементы ссылаются на нулевые указатели.

1. **createMCList**

Опишем функцию для создания связанного списка.

MusicalComposition\* createMusicalCompositionList(char\*\* array\_names, char\*\* array\_authors, int\* array\_years, int n)

{

MusicalComposition\* hold = NULL;

MusicalComposition\* temp = NULL;

MusicalComposition\* start=NULL;

start = createMusicalComposition(array\_names[0], array\_authors[0], array\_years[0]);

hold = start;

int i;

for (i = 1; i<n; i++)

{

temp = start;

start->next = createMusicalComposition(array\_names[i], array\_authors[i], array\_years[i]);

start = start->next;

start->prev = temp;

}

start = hold;

return start;

}

Первый элемент создается отдельно, а последующие с помощью цикла. Цикл выглядит так: сохранение текущего элемента, создание следующего и переход на него и присваивание раннее сохраненного значения в качестве предыдущего. Данный цикл выполняется n-1 раз, так как первый элемент уже создан. Конечные элементы ссылаются на нулевые указатели и «ограждают» список.

1. **push**

Функция для добавление n элементов в конец списка

void push(MusicalComposition\* head)

{

int i, k;

MusicalComposition\* end = NULL;

printf("Количество добавляемых элементов в конец: ");

k = atoi(get\_name());

char\*\* names = (char\*\*)malloc(sizeof(char\*)\*k);

char\*\* authors = (char\*\*)malloc(sizeof(char\*)\*k);

int\* years = (int\*)malloc(sizeof(int)\*k);

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

printf("[%d] Композиция: ", i+1);

names[i] = get\_name();

printf("[%d] Автор: ", i+1); //Заполняем массивы строк и чисел

authors[i] = get\_name();

printf("[%d] Год: ", i+1);

years[i] = atoi(get\_name());

}

end = createMusicalCompositionList(names, authors, years, k);

while(head->next)

{ //Передвигаемся по основному списку head = head->next; //до последнего элемента

}

head->next = end;

end->prev = head;

}

Сначала считывается количество добавляемых элементов, затем функцией **malloc** выделяется память под массив указателей на строки, содержащие названия композиций и авторов групп, и под массив из целых чисел. С помощью цикла и созданной раннее функции **get\_name** заполняем массивы строк и чисел (для чисел используем функцию **atoi,** т.к. **get\_name** возвращает указатель на символ). Далее создается «мини-список» **end**, который свзяывается с помощью указателей с основным списком.

1. **removeEl**

Опишем функцию для удаления элемента по имени композиции

void removeEl(MusicalComposition\*\* TrueHead, char\* name\_for\_remove)

{

MusicalComposition \*head = \*TrueHead;

while(head)

{

if (strcmp(head->name, name\_for\_remove)==0)

{

if(head->next == NULL && head->prev == NULL){

\*TrueHead = NULL;

free(head);

return;

} else if (head->next == NULL){

head->prev->next = NULL;

free(head);

return;

} else if (head->prev == NULL){

head->next->prev = NULL;

\*TrueHead = head->next;

free(head);

return;

} else {

head->next->prev = head->prev;

head->prev->next = head->next;

free(head);

return;

}

}

head = head->next;

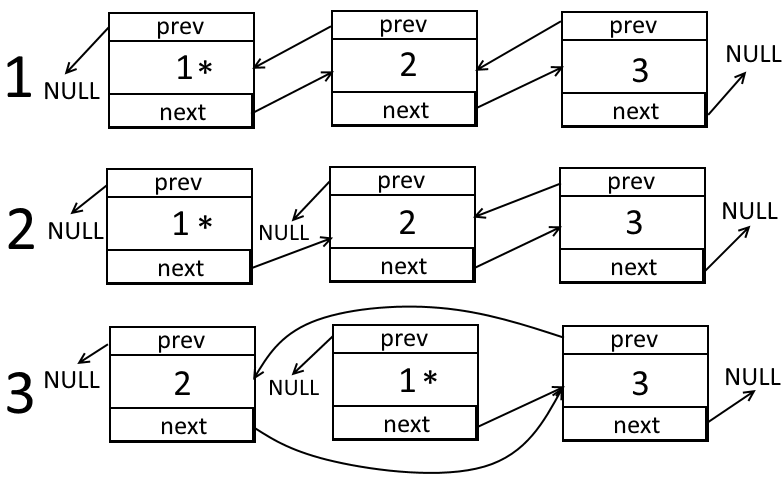
}

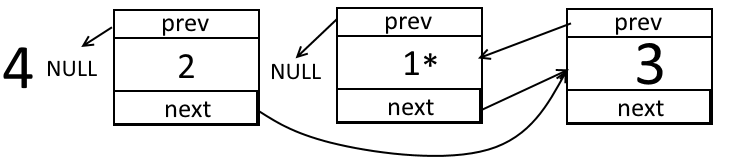
printf("\nВ списке нет данной композиции!\n");

}

Цикл выполняется, пока элемент не примет значение **NULL.** Данный цикл прозводит поиск элемента с соответствующим именем композиции, и если такой элемент найдется, то возможны следующие варианты:

* Элемент является единственным в списке: в этом случае указатель на голову принимает значение **NULL**. В дальнейшем, в функции меню будет проверятся является ли указатель нулевым. Если да, то программа будет сообщать, что список отсутствует, т.е. список удален полностью.
* Элемент является последним: тогда в качестве следующего элемента для предыдущего будет указатель на **NULL,** т. е. список обрывается
* Элемент стоит посередине: в этом случае переставляются указатели на связанные элементы у окружающих элементов.
* Элемент является первым: в этом случае предыдущий элемент следующего будет ссылаться на **NULL** (2 на схеме)**.**  Затем «голова» принимает значения второго элемента, который теперь становится «головой» (3). На схеме:





Если цикл выполнился полностью без прерываний, то выдается сообщение, что введеного элемента в списке нет

1. **count**

Опишем функцию, подсчитывающую количество элементов в списке

int count(MusicalComposition\* head)

{

int count=0;

while(head)

{

count++;

head = head->next;

}

return count;

}

Пока голова не примет значение **NULL,** цикл выполняется и увеличивает счетчик.

1. **AddToBegin**

Опишем функцию, добавляющую в начало списка n элементов.

MusicalComposition\* AddToBegin(MusicalComposition\* head)

{

int i, k;

printf("Количество добавляемых элементов в начало: ");

k = atoi(get\_name());

char\*\* names = (char\*\*)malloc(sizeof(char\*)\*k);

char\*\* authors = (char\*\*)malloc(sizeof(char\*)\*k);

int\* years = (int\*)malloc(sizeof(int)\*k);

for (i=0;i<k;i++)

{

printf("[%d] Композиция: ", i+1);

names[i] = get\_name();

printf("[%d] Автор: ", i+1);

authors[i] = get\_name();

printf("[%d] Год: ", i+1);

years[i] = atoi(get\_name());

}

MusicalComposition\* NewHead;

NewHead= createMusicalCompositionList(names, authors, years, k);

MusicalComposition\* temp = NewHead;

while(NewHead->next){

NewHead = NewHead->next; //Передвигаемся по «мини-списку» до последнего элем

}

NewHead->next = head;

head->prev = NewHead;

return temp;

}

По аналогии с **push** считываем количество добавляемых элементов, выделяем память под массив указателей и массив чисел затем заполняем массив строк функцией **get\_name.** Далее создаем «мини-список» и связываем его с основным.

1. **removeYear**

Опишем функцию, которая удаляет все элементы с годом меньше заданного

MusicalComposition\* removeYear(MusicalComposition\* head)

{

MusicalComposition \*i, \*start = head;

printf("Удалить элементы, чей год меньше: ");

int k = atoi(get\_name());

while(head){

if (head->year < k){

if (head->next ==NULL && head->prev == NULL){

free(head);

return NULL;

}

if (head->next == NULL){

head->prev->next = NULL;

free(head);

break;

} else if (head->prev == NULL){

head->next->prev = NULL;

start = head->next;

free(head);

} else {

head->next->prev = head->prev;

head->prev->next = head->next;

free(head);

}

}

head = head->next;

}

return start;

}

Данная функция в отличие от **removeEl** возвращает указатель на структуру, поэтому имеются некоторые отличия. Цикл **while** выполняет поиск элементов с годом меньше указанного. Если такой элемент найден, то возможны следующие варианты:

* Элемент является единственным в списке: в этом случае возвращается **NULL** и в функции-меню предусмотрено, если указатель на структуру равен нулю, то программа будет сообщать, что список отсутствует.
* Элемент является последним: тогда в качестве следующего элемента для предыдущего будет указатель на **NULL,** т. е. список обрывается
* Элемент стоит посередине: в этом случае переставляются указатели на связанные элементы у окружающих элементов.
* Элемент является первым: в этом случае предыдущий элемент следующего будет ссылаться на **NULL.** Затем «головой» становится следующая структура. В конце функции возвращается новая голова.

1. **print\_names**

Создадим функцию, выводящую текущий список в виде таблицы. Для этого будем использовать символы псевдографики.

void print\_names(MusicalComposition\* head)

{

int i = 1;

printf("┌─────┬──────────────────────────┬─────────────────────┬─────┐\n");

printf("│Номер│ Композиция│ Исполнитель│ Год│\n");

printf("╞═════╪══════════════════════════╪═════════════════════╪═════╡\n");

while(head)

{

printf("│%5d│%26s│%21s│%5d│\n", i++, head->name, head->author, head->year);

if (head->next == NULL) printf("└─────┴──────────────────────────┴─────────────────────┴─────┘\n");

else printf("├─────┼──────────────────────────┼─────────────────────┼─────┤\n");

head = head->next;

}

};

Сначала формируется «голова» таблицы, затем с помощью цикла выводится информация о всех элементах списка. Если указатель на следующий является нулевым, то таблица закрывается.

1. **menu**

Создадим функцию-меню для работы со списком. Полный код с комментариями находится в *Приложении 1.*

1. **func.h**

Создадим заголовочный файл, в который поместим прототипы всех функций:

void menu();

char\* get\_name();

MusicalComposition\* createMusicalComposition(char\*, char\*, int);

MusicalComposition\* createMusicalCompositionList(char\*\*, char\*\*, int\*, int);

void push(MusicalComposition\*);

void removeEl(MusicalComposition\*\*, char\*);

int count(MusicalComposition\*);

void print\_names(MusicalComposition\*);

MusicalComposition\* AddToBegin(MusicalComposition\*);

MusicalComposition\* removeYear(MusicalComposition\*);

Работа с “make” и сборка проекта

1. Для удобства сборки проекта создадим **makefile** следующего содержания:

List: main.o func.o struct.h func.h

gcc main.o -o List.out func.o

rm \*.o

main.o: main.c struct.h func.h

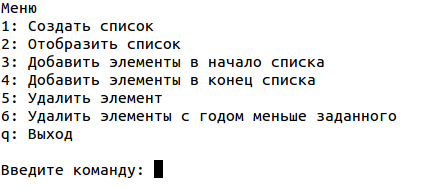
gcc -c main.c

func.o: func.c struct.h func.h

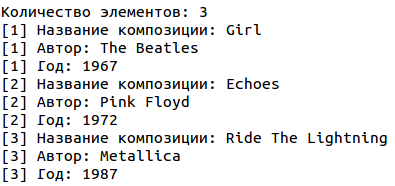
gcc -c func.c

* Сначала создаются два объектных файла **main.o** и **func.o** : Препроцессор объединяет исходный код файлов с кодом из заголовочных файлов стандартных библиотек языка Си, а также с кодом из локальных заголовочных файлов, в которых хранятся прототипы функций и объявление структуры. Затем из объедененного кода формируются объектные файлы.
* Линковщик производит линковку объектных файлов и формирует из них единый исполняемый файл. Затем промежуточные объектные файлы удаляются.

1. Соберем программу, используя комманду make List.
2. Протестируем работоспособность программы .
   1. Запуск и меню



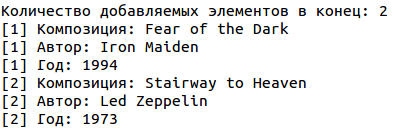
* 1. Нажмем 1 и создадим список



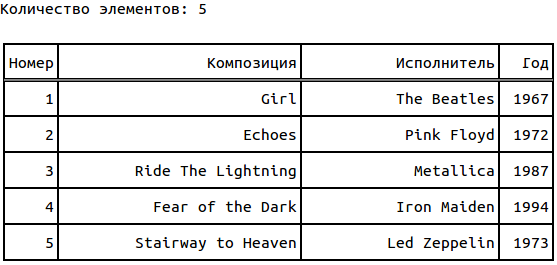
* 1. В меню нажмем 2 и выведем полученный список



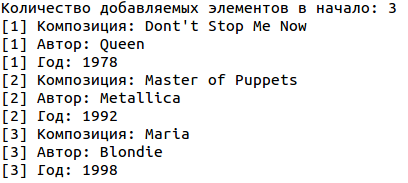
* 1. Нажмем 4 и добавим два элемента в конец списка



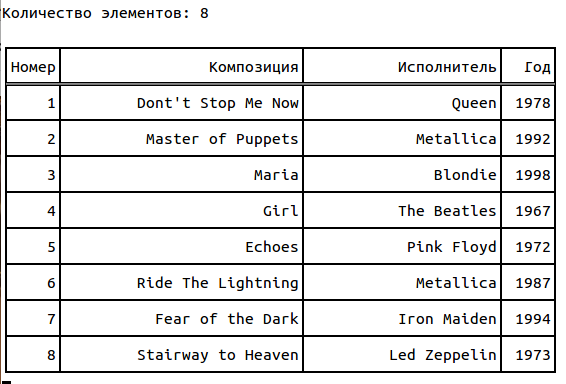
* 1. Снова выведем текущий список



* 1. Нажмем 3 и добавим три элемента в начало списка.



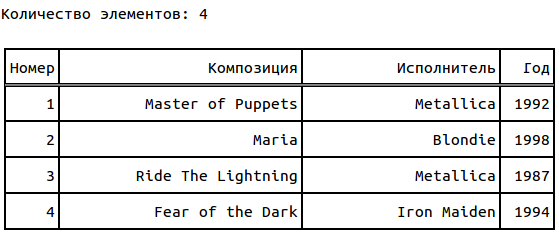
* 1. Отобразим вновь наш список



* 1. Удалим композицию группы “The Beatles” “Girl” и выведем список вновь.



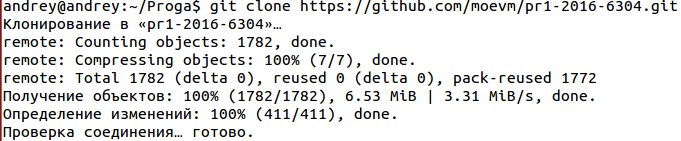
* 1. Теперь удалим все композиции, чей год меньше 1980, и выведем на экран полученный результат.



Во всех случаях программа отработала корректно.

Работа с “github”

1. Клонируем репозиторий группы



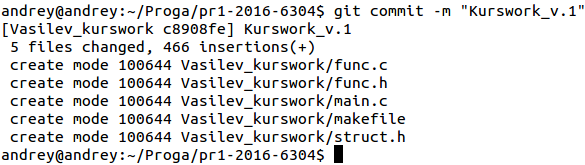
1. Создадим новую ветку командой

git checkout –b “Vasilev\_kurswork”

1. Копируем папку **Vasilev\_kurswork** в репозиторий
2. Добавляем папку командой

git add Vasilev\_kurswork

1. Описываем изменения командой



1. Пушим ветку в репозиторий командой

git push origin Vasilev\_kurswor

ЗАКЛЮЧЕНИЕ

Выполнив данную работу, мы закрепили на практике знания о структурах, создав связанный двунаправленный список, состоящий из структур. Мы научились создавать API для работы с такими списками. Также мы попрактиковались в работе с проектом, состоящим из нескольких частей: создали заголовки и файлы с кодом, создали меню для удобной работы с созданным API и создали makefile для удобного обращения с проектом.

ПРИЛОЖЕНИЕ 1

void menu()

{

char\* bufClean = NULL; //Эта переменная нужна для задержки экрана

system("clear"); //Функция для очищения экрана

int size;

char\* name = NULL;

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

MusicalComposition\* head = NULL;

int length;

int i;

int ListCreated = 0; //Есть ли сейчас список в памяти.

char\* choice = "m"; //Строка для выбора в меню

while(choice[0] != 'q'){

printf("Меню\n1: Создать список \n2: Отобразить список \n3: Добавить элементы в начало списка \n");

printf("4: Добавить элементы в конец списка \n5: Удалить элемент\n6: Удалить элементы с годом меньше заданного\nq: Выход\n\nВведите команду: ");

choice = get\_name();

system("clear"); //Получаем команду для выполнения

switch(choice[0]){

case '1':

//Если список уже создан, то спрашиваем, хочет ли пользователь записать новый список

if(ListCreated){

printf("\n\n\n\nВаш старый список будет удален! Продолжить? (y/n): ");

choice = get\_name();

if (choice[0] == 'y') printf("\n");

else {

system("clear");

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

break;

}

}

printf("\n\n\n\nКоличество элементов: ");

length = atoi(get\_name());

char\*\* names = (char\*\*)malloc(sizeof(char\*)\*length);

char\*\* authors = (char\*\*)malloc(sizeof(char\*)\*length);

int\* years = (int\*)malloc(sizeof(int)\*length);

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

{

printf("[%d] Название композиции: ", i+1);

names[i] = get\_name(); //Заполняем массив

printf("[%d] Автор: ", i+1); //массив из строк

authors[i] = get\_name(); //и массив из

printf("[%d] Год: ", i+1); //символов

years[i] = atoi(get\_name());

}

head = createMusicalCompositionList(names, authors, years, length); //Создаем список функцией, описанной выше

ListCreated = 1; //Список создан

system("clear");

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

break;

case '2':

//Если список не создан, то оповещаем об этом пользователя и возвращаемся в меню

if (ListCreated == 0){

printf("\n\n\n\nСписок не создан\n");

bufClean = get\_name();

system("clear");

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

break;

}

printf("\n\nКоличество элементов: %d\n\n", count(head));

print\_names(head);

bufClean = get\_name(); //Выводим таблицу со

system("clear"); //списком и количество

printf("\n\n\n\n"); //элементов

break;

case '3':

//Если список не создан, то оповещаем об этом пользователя и возвращаемся в меню

if (ListCreated == 0){

printf("\n\n\n\nСписок не создан!\n");

bufClean = get\_name();

system("clear");

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

break;

}

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

head = AddToBegin(head); //Вызываем функцию добавления

system("clear"); //в начало

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

break;

case '4':

//Если список не создан, то оповещаем об этом пользователя и возвращаемся в меню

if (ListCreated == 0){

printf("\n\n\n\nСписок не создан!\n");

bufClean = get\_name();

system("clear");

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

break;

}

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

push(head); //Вызываем функцию добавления в конец

system("clear");

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

break;

case '5':

//Если список не создан, то оповещаем об этом пользователя и возвращаемся в меню

if (ListCreated == 0){

printf("\n\n\n\nСписок не создан!\n");

bufClean = get\_name();

system("clear");

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

break;

}

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

print\_names(head);

printf("ВВедите имя элемента, который хотите удалить: ");

size = count(head); //Размер до удаления

name = get\_name(); //Вводим имя для удаления

removeEl(&head, name); //Вызываем функцию удаления

/\*Если указатель на начало списка равен **NULL** , то оповещаем что удален последний элемент списка. Также ставим счетчик того, создан ли лист на 0\*/

if(head == NULL) {

printf("\nУдален последний элемент!\n");

ListCreated = 0;

} else if (size != count(head)){

/\*Если количество до и после неравно, то оповещаем, что элемент успешно удален\*/

printf ("\nЭлемент удален!\n");

/\*Если количество до и после равно, то оповещаем, что элемент не удален (скорее всего введено неверное имя для удаления\*/

} else printf("\nЭлемент не удален!\n");

bufClean = get\_name();

system("clear");

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

break;

case '6':

//Если список не создан, то оповещаем об этом пользователя и возвращаемся в меню

if (ListCreated == 0){

printf("\n\n\n\nСписок не создан!\n");

bufClean = get\_name();

system("clear");

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

break;

}

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

print\_names(head); //Выводим таблицу

head = removeYear(head); //Вызываем функцию удаления

/\*Если голова ссылается на “NULL”, то оповещаем, что удален последний элемент (NULL присваивается в функции removeYear, когда элемент один). Присваиваем ListCreated значение 0, программа теперь считает, что списка нет\*/

if (head == NULL){

printf("\nУдален последний элемент списка!\n");

bufClean = get\_name();

ListCreated = 0;

}

system("clear");

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

break;

case 'q':

//Если список создан, то спрашиваем, хочет ли пользователь выйти и потерять список

if(ListCreated){

printf("\n\n\n\nВаш список будет утерян! Продолжить? (y/n): ");

choice = get\_name();

if (choice[0] == 'y') printf("\n");

else {

system("clear");

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

break;

}

}

choice[0] = 'q';

printf("\n\nBye\n\n\n");

break;

//Если введена команда, которой нет в меню, то выводим сообщение об ошибке

default: printf("\n\nОшибка, такой команды нет.\n\n");

}

}

}