МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МОЭВМ

КУРСОВАЯ РАБОТА

по дисциплине «Программирование»

Тема: «Линейные списки»

студент гр. 7381 _____ Трушников А.П

подаватель _____ Берленко Т.А

Санкт-Петербург
2017

АННОТАЦИЯ

В данной работе была разработана программа на языке программирования С, которая позволяет работать с набором функций, отвечающих за список музыкальных композиций. Для функционирования списка были созданы и описаны необходимые функции, позволяющие добавлять элемент в список, удалять элемент соответствующего имени, сортировать список по возрастанию года, выводить элементы списка и их количество в консоль. В данной работе приведено полное описание исходного кода, результаты работы программы.

СОДЕРЖАНИЕ

ЗАДАНИЕ НА КУРСОВУЮ РАБОТУ

Требуется написать Application Program Interface для работы с двунаправленным линейным списком MusicalComposition, написать функцию, которая добавляет элементы в список, удаляет их, сортирует элементы списка по возрастанию, может удалять начало чписка, выводит список, считает количество элементов в списке, очищает список.

ВВЕДЕНИЕ

Цель курсовой работы – изучить понятия структуры в языке программирования С, изучить структуру данных, именуемую списком (одно- и двусвязный): что это такое, для чего, доступные операции для неё, их асимптотическая сложность. Результатом выполнения курсовой работы должен быть проект, содержащий в себе функции, позволяющие добавлять элемент в список, удалять элемент соответствующего имени, сортировать список по убыванию года, выводить элементы списка и их количество в консоль. В данной работе приведено полное описание исходного кода, результаты работы программы.

1.Описание структуры struct MusicalComposition.

struct MusicalComposition – структура, экземпляры которой являюся элементами двусвязного списка. Содержащая в себе поля музыкальной композици, а также указатели на соседние struct MusicalComposition.

Тип	Имя	Описание
char	name[80]	Имя музыкальной композиции
char	author[80]	Автор музыкальной композиции
int	year	Год публикации музыкальной композиции
struct MusicalComposition*	next	Указатель на следующую музыкальную композицию
struct MusicalComposition*	prev	Указатель на предыдущую музыкальную композицию

Аргументы:

char name[80] -строка, имя конструируемой музыкальной композиции.

char author[80] -строка, автор конструируемой музыкальной композиции.

int year -целое число, год конструируемой музыкальной композиции.

struct MusicalComposition* next -указатель на следующий сконструированный экземпляр композиции.

struct MusicalComposition* prev -указатель на предыдущий сконструированный экземпляр композиции

2.Описание функций для работы с двунаправленным линейным списком.

2.1Создание элемента списка.

MusicalComposition* createMusicalComposition(char* name, char* autor,int year);

Функция принимает в качестве аргументов указатели на название композиции (char* name) и ее автора (char* author), а также год написания (int year). Поля заполняются значениями. Функция возвращает указатель.

Возвращаемое значение – указатель на созданный элемент.

Код функции можно посмотреть в приложении А.

2.2Создание двунаправленного списка.

MusicalComposition* createMusicalCompositionList(MusicalComposition* head,char** array_names, char** array_authors, int* array_years, int n);

Данная функция создает список музыкальных композиций. В качестве аргументов функция принимает указатель на голову списка (MusicalComposition* head), указатель на указатель на массив с названиями композиций (char** array_names), на массив с именами авторов (char** array_authors), указатель на массив с годами написания композиций (int* array_years) и размер массива (int n). Сначала создается “голова” списка и заполняется данными. Далее по циклу запускается заполнение полей и установление взаимосвязей между элементами списка. Функция возвращает указатель на «голову» списка.

Возвращаемое значение – указатель на “голову” списка.

Код функции можно посмотреть в приложении Б.

2.3Добавление элемента в список.

MusicalComposition* push(MusicalComposition* head, MusicalComposition* element);

Данная функция добавляет новый элемент в список. В качестве аргументов функция принимает указатель на «голову» списка (MusicalComposition* head) и указатель на элемент, которые необходимо добавить в конец списка (MusicalComposition* element).

Возвращаемое значение – указатель на “голову” списка.

Код функции можно посмотреть в приложении В.

2.4Удаление элемента из списка.

MusicalComposition* removeEl(MusicalComposition* head, char* name_for_remove);

Данная функция удаляет из списка элемент, имя которого совпадает с именем для удаления. В качестве аргументов функция принимает указатель на «голову» списка (MusicalComposition* head) и название композиции, которую необходимо удалить (char* name_for_remove). Для каждого его элемента происходит сравнение названия композиции с именем произведений, которые нужно удалить.

Возвращаемое значение – указатель на “голову” списка.

Код функции можно посмотреть в приложении Г.

2.5Подсчет количества элементов в списке.

int count(MusicalComposition* head);

Данная функция подсчитывает количество элементов в функции. В качестве аргумента она принимает указатель на «голову» списка (MusicalComposition* head).

Возвращаемое значение –количество элементов списка.

Код функции можно посмотреть в приложении Д.

2.6Вывод списка.

void print_names(MusicalComposition* head);

Функция выводит на экран названия композиций, их авторов и год создания в списке. В качестве аргумента функция принимает указатель на «голову» списка (MusicalComposition* head).

Возвращаемое значение – нет.

Код функции можно посмотреть в приложении Е.

2.7Смена двух элементов местами.

MusicalComposition* swap_list(MusicalComposition* head, char* a ,char* b);

В качестве аргументов данная функция принимает указатели на голову и на названия песен, которые необходимо поменять местами (MusicalComposition* head, char* a ,char* b). Функция swap_list меняет местами два элемента списка путем изменения их указателей на следующий и предыдущий элементы, а также изменения указателей у предыдущего для первого элемента и следующего для второго. Функция учитывает случаи, когда имеется всего два элемента, которые необходимо поменять. Случай, когда необходимо поменять «голову» списка со следующим элементом. Случай, когда меняются два соседние элемента в середине списка. Случай, когда необходимо поменять местами два последних элемента списка.

Возвращаемое значение . Если элементы менялись местами в конце списка, то указатель на “голову”. Если элементы менялись местами в конце списка, то указатель на элемент, у которого имя принимается функцией в качестве 3 аргумента. Если элементы менялись местами в середине списка, то указатель на “голову”.

Код функции можно посмотреть в приложении Ж.

2.8Сортировка по году, по возрастанию.

MusicalComposition* sort(MusicalComposition* head);

Функция sort выполняет сортировку списка по возрастанию года создания музыкальной композиции. Данная функция в качестве аргумента принимает указатель на «голову» списка (MusicalComposition* head). В цикле проверяется условие: если год создания первого элемента больше второго, то они меняются местами.

Возвращаемое значение указатель на “голову” списка.

Код функции можно посмотреть в приложении З.

2.9Удаление “головы” списка.

MusicalComposition* head_del(MusicalComposition* head);

Функция head_del выполняет удаление “головы” списка. Данная функция в качестве аргумента принимает указатель на «голову» списка (MusicalComposition* head).

Возвращаемое значение указатель на следующий элемент списка.

Код функции можно посмотреть в приложении И.

2.10Удаление “хвоста” списка.

MusicalComposition* tail_del(MusicalComposition* head);

Функция tail_del выполняет удаление “головы” списка. Данная функция в качестве аргумента принимает указатель на «голову» списка (MusicalComposition* head).

Возвращаемое значение указатель на “голову” списка.

Код функции можно посмотреть в приложении К.

2.11Поиск элемента списка.

MusicalComposition* find(char* a,MusicalComposition* head);

Функция find выполняет поиск элемента списка. Данная функция в качестве аргумента принимает указатель на «голову» списка (MusicalComposition* head) и на название музыкальной композиции (char* a).

Возвращаемое значение указатель на “голову” списка.

Код функции можно посмотреть в приложении Л.

2.12Добавление первого элемента списка.

MusicalComposition* tail_add(MusicalComposition* head,MusicalComposition* list);

Данная функция добавляет новый элемент в начало списка. В качестве аргументов функция принимает указатель на «голову» списка (MusicalComposition* head) и указатель на элемент, которые необходимо добавить в начало списка (MusicalComposition* list).

Возвращаемое значение – указатель на элемент, который был ,добавлен.

Код функции можно посмотреть в приложении М.

2.13Добавление элемента списка после указанного.

MusicalComposition* add_elem(MusicalComposition* head,MusicalComposition* list,char* a);

Данная функция добавляет новый элемент после элемента, в котором название музыкальной композиции совпадает с 2 принимаемым аргументом функции (MusicalComposition* list,char* a). В качестве аргументов функция принимает указатель на «голову» списка (MusicalComposition* head) и указатель на элемент, которые необходимо добавить (MusicalComposition* list), и на строку, которая должна совпадать с названием музыкальной композиции после которой мы добавляем элемент (char* a).

Возвращаемое значение – указатель на “голову” списка.

Код функции можно посмотреть в приложении Н.

2.14Очистка списка.

MusicalComposition* clean_list(MusicalComposition* head);

Функция освобождает список музыкальных композиций. В качестве аргумента она принимает указатель на “голову” списка.

Возвращаемое значение – NULL.

Код функции можно посмотреть в приложении О.

2.15Функция main.

```
int main( );
```

Функция main показывает меню, с несколькими вариантами обработки списка. Данная функция в качестве аргумента ничего не принимает.

Возвращаемое значение - 0.

Код функции можно посмотреть в приложении П.

3.Проверка работы программы

3.1Тест №1.

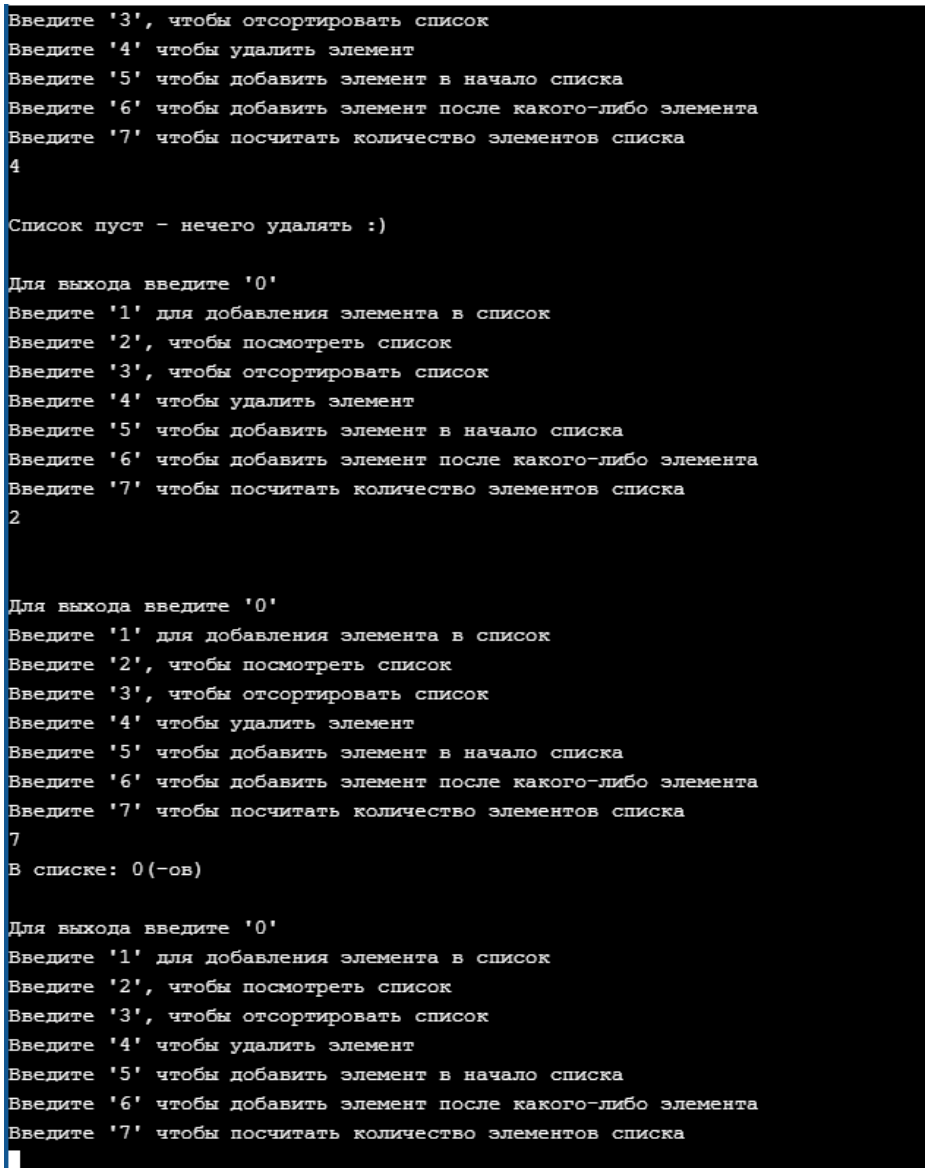

Рис.1 – демонстрация выполнения 1-ого теста.

Попытка удаления музыкальной композиции из пустого списка не приводит к завершению работы программы. Вывод пустого списка не производится. Количество элементов в пустом списке определено верно и равно нулю. Освобождение памяти для пустого списка при выходе из программы производится корректно, не происходит ошибки segmentation fault(см. рис. 1).

3.2Тест №2.

```
Для выхода введите '0'
Введите '1' для добавления элемента в список
Введите '2', чтобы посмотреть список
Введите '3', чтобы отсортировать список
Введите '4' чтобы удалить элемент
Введите '5' чтобы добавить элемент в начало списка
Введите '6' чтобы добавить элемент после какого-либо элемента
Введите '7' чтобы посчитать количество элементов списка
5
Введите название музыкальной композиции:Sonne1
Введите автора музыкальной композиции:asdsa
Введите год публикации музыкальной композиции:1999
Для выхода введите '0'
Введите '1' для добавления элемента в список
Введите '2', чтобы посмотреть список
Введите '3', чтобы отсортировать список
Введите '4' чтобы удалить элемент
Введите '5' чтобы добавить элемент в начало списка
Введите '6' чтобы добавить элемент после какого-либо элемента
Введите '7' чтобы посчитать количество элементов списка
5
Введите название музыкальной композиции:Sonne2
Введите автора музыкальной композиции:sadsa
Введите год публикации музыкальной композиции:1999
Для выхода введите '0'
Введите '1' для добавления элемента в список
Введите '2', чтобы посмотреть список
Введите '3', чтобы отсортировать список
Введите '4' чтобы удалить элемент
Введите '5' чтобы добавить элемент в начало списка
Введите '6' чтобы добавить элемент после какого-либо элемента
Введите '7' чтобы посчитать количество элементов списка
1
```

Рис.2 – демонстрация выполнения 2-ого теста.

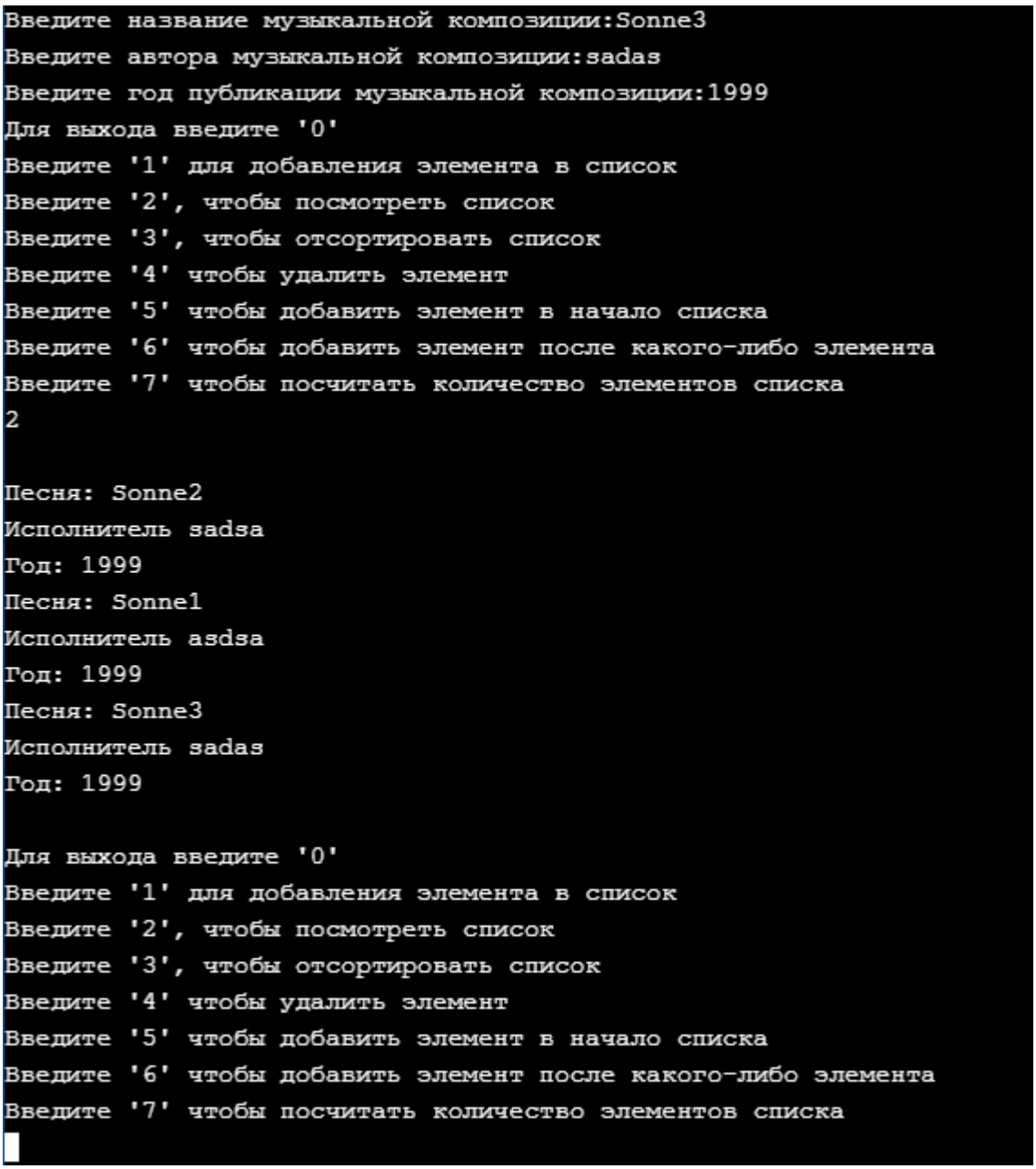

Рис.3 – демонстрация выполнения 2-ого теста.

```
4
Введите название музыкальной композиции, чтобы удалитSonne2
Для выхода введите '0'
Введите '1' для добавления элемента в список
Введите '2', чтобы посмотреть список
Введите '3', чтобы отсортировать список
Введите '4' чтобы удалить элемент
Введите '5' чтобы добавить элемент в начало списка
Введите '6' чтобы добавить элемент после какого-либо элемента
Введите '7' чтобы посчитать количество элементов списка
2

Песня: Sonne1
Исполнитель asdsa
Год: 1999
Песня: Sonne3
Исполнитель sadas
Год: 1999

Для выхода введите '0'
Введите '1' для добавления элемента в список
Введите '2', чтобы посмотреть список
Введите '3', чтобы отсортировать список
Введите '4' чтобы удалить элемент
Введите '5' чтобы добавить элемент в начало списка
Введите '6' чтобы добавить элемент после какого-либо элемента
Введите '7' чтобы посчитать количество элементов списка
7
В списке: 2(-ов)

Для выхода введите '0'
Введите '1' для добавления элемента в список
Введите '2', чтобы посмотреть список
Введите '3', чтобы отсортировать список
Введите '4' чтобы удалить элемент
Введите '5' чтобы добавить элемент в начало списка
Введите '6' чтобы добавить элемент после какого-либо элемента
Введите '7' чтобы посчитать количество элементов списка
```

Рис.4 – демонстрация выполнения 2-ого теста.

Добавление двух музыкальных композиций в начало списка и один в конец. Функция определения количества музыкальных композиций в списке отработала корректно (три композиции). Вывод информации обо всех музыкальных композициях также произведён в правильном порядке, согласно указанному

порядку вставки с учётом вставки в начало- конец списка. Удаление одной композиции из начала списка. Вывод количества оставшихся композиций в списке (две), а также информации об этой композиции. Завершение работы приложения, освобождение памяти из-под оставшейся музыкальной композиции. Segmentation fault отсутствует.(см. рис.2 , рис.3, рис.4)

3.3Тест №3.

```
Песня: Sonne1
Исполнитель dfds
Год: 1999
Песня: Sonne2
Исполнитель sadas
Год: 1999

Для выхода введите '0'
Введите '1' для добавления элемента в список
Введите '2', чтобы посмотреть список
Введите '3', чтобы отсортировать список
Введите '4' чтобы удалить элемент
Введите '5' чтобы добавить элемент в начало списка
Введите '6' чтобы добавить элемент после какого-либо элемента
Введите '7' чтобы посчитать количество элементов списка
6
Введите название музыкальной композиции, после которой хотите добавить элемент:Sonne1
Введите название музыкальной композиции, которую хотите добавить:Sonne4
Введите автора музыкальной композиции, которую хотите добавить:sadsa
Введите год публикации музыкальной композиции, которую хотите добавить:1999
Для выхода введите '0'
Введите '1' для добавления элемента в список
Введите '2', чтобы посмотреть список
Введите '3', чтобы отсортировать список
Введите '4' чтобы удалить элемент
Введите '5' чтобы добавить элемент в начало списка
Введите '6' чтобы добавить элемент после какого-либо элемента
Введите '7' чтобы посчитать количество элементов списка
6
```

Рис.5 – демонстрация выполнения 3-ого теста.

```
Введите название музыкальной композиции, после которой хотите добавить элемент:Sonne1
Введите название музыкальной композиции, которую хотите добавить:Sonne6
Введите автора музыкальной композиции, которую хотите добавить:dsadsa
Введите год публикации музыкальной композиции, которую хотите добавить:1999
Для выхода введите '0'
Введите '1' для добавления элемента в список
Введите '2', чтобы посмотреть список
Введите '3', чтобы отсортировать список
Введите '4' чтобы удалить элемент
Введите '5' чтобы добавить элемент в начало списка
Введите '6' чтобы добавить элемент после какого-либо элемента
Введите '7' чтобы посчитать количество элементов списка
2

Песня: Sonne1
Исполнитель dfds
Год: 1999
Песня: Sonne6
Исполнитель dsadsa
Год: 1999
Песня: Sonne4
Исполнитель sadsa
Год: 1999
Песня: Sonne2
Исполнитель sadas
Год: 1999

Для выхода введите '0'
Введите '1' для добавления элемента в список
Введите '2', чтобы посмотреть список
Введите '3', чтобы отсортировать список
```

Рис.6 – демонстрация выполнения 3-ого теста.

```
Песня: Sonne1
Исполнитель asdsa
Год: 1999
Песня: Sonne4
Исполнитель sadas
Год: 1999
Песня: Sonne5
Исполнитель sadsa
Год: 1999
Песня: Sonne2
Исполнитель dsadsa
Год: 1999

Для выхода введите '0'
Введите '1' для добавления элемента в список
Введите '2', чтобы посмотреть список
Введите '3', чтобы отсортировать список
Введите '4' чтобы удалить элемент
Введите '5' чтобы добавить элемент в начало списка
Введите '6' чтобы добавить элемент после какого-либо элемента
```

Рис.7 – демонстрация выполнения 3-ого теста.

Использование функции вставки двух музыкальных композиций в середину списка музыкальных композиций, вывод списка. Segmentation fault отсутствует.(см. рис 5, см. рис.6, см. рис. 7)

3.2Тест №4

```
Введите название музыкальной композиции, после которой хотите добавить элемент:Sonne1
Введите название музыкальной композиции, которую хотите добавить:Sonne6
Введите автора музыкальной композиции, которую хотите добавить:dsadsa
Введите год публикации музыкальной композиции, которую хотите добавить:1999
Для выхода введите '0'
Введите '1' для добавления элемента в список
Введите '2', чтобы посмотреть список
Введите '3', чтобы отсортировать список
Введите '4' чтобы удалить элемент
Введите '5' чтобы добавить элемент в начало списка
Введите '6' чтобы добавить элемент после какого-либо элемента
Введите '7' чтобы посчитать количество элементов списка
2

Песня: Sonne1
Исполнитель dfds
Год: 1999
Песня: Sonne6
Исполнитель dsadsa
Год: 1999
Песня: Sonne4
Исполнитель sadsa
Год: 1999
Песня: Sonne2
Исполнитель sadas
Год: 1999

Для выхода введите '0'
Введите '1' для добавления элемента в список
Введите '2', чтобы посмотреть список
Введите '3', чтобы отсортировать список
```

.

Рис.8 – демонстрация выполнения 4-ого теста.

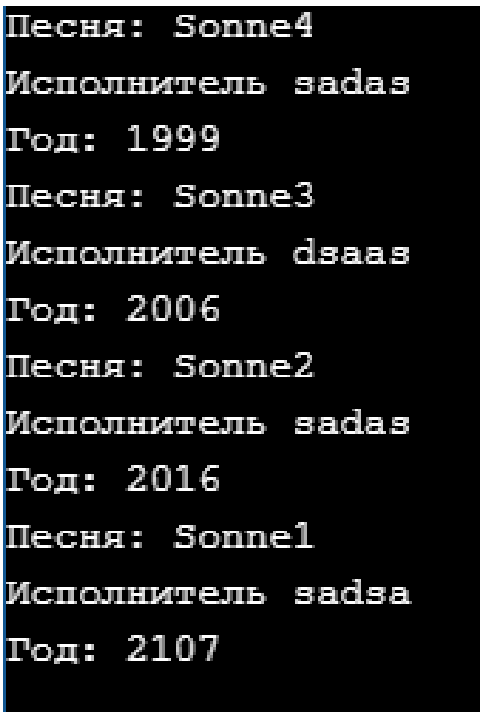

Рис.9 – демонстрация выполнения 4-ого теста.

Использование функции сортировки списка музыкальных композиций по году по возрастанию, вывод списка. Segmentation fault отсутствует.(см. рис 8, см. рис.9)

ВЫВОД

В ходе выполнения курсовой работы, были закреплены навыки программирования на языке C, были изучены понятие структуры struct в языке C, структура данных двусвязный список, её поддерживаемый операции (вставка, удаление, поиск), их асимптотика (вставка и удаление элемента в списке производится за константное время, однако, оно может быть и линейным, если точно не известен адрес удаляемого элемента или элемента, возле которого необходимо произвести вставку. Связано это с тем, что поиск в двусвязном списке осуществляется за линейное время). Были закреплены навыки работы с утилитой make, а также системой контроля версий github.

ПРИЛОЖЕНИЯ

Приложение А.

```
MusicalComposition* createMusicalComposition(char* name, char* autor,int year){//ghghgghhg
   MusicalComposition* mus=(MusicalComposition*)malloc(sizeof(MusicalComposition));
   int i;
   for(i=0;i<80;i++){
      mus->name[i]=name[i];
      mus->author[i]=autor[i];
   }
   mus->year=year;
   mus->next=NULL;
   mus->prev=NULL;
   return mus;
}
```

Приложение Б.

```
MusicalComposition* createMusicalCompositionList(MusicalComposition* head,char* array_names, char* array_authors, int array_years){
      MusicalComposition* mus=createMusicalComposition(array_names,array_authors,array_years);
      head = push(head,mus);
   return head;
}
```

Приложение В.

```
MusicalComposition* push(MusicalComposition* head, MusicalComposition* element){//fdfdffdfd
    if(head==NULL){
        return element;
    }

    if(head->next==NULL){
        head->next=element;
        element->prev=head;
        return head;
    }
    push(head->next,element);

    return head;
}
```

Приложение Г.

```
MusicalComposition* removeEl(MusicalComposition* head, char* name_for_remove){//sdsdsdsd
    int i;  //gfdfgdfgfd
    int equal=1;
    if(head==NULL){
        return NULL;
    }
```

```
for(i=0;i<80;i++){
    if(head->name[i]=='\0')
        break;
    if(head->name[i]!=name_for_remove[i]){
        equal=0;
    }
}
if(equal==1){
    if(head->next==NULL&&head->prev==NULL){
        return NULL;
    }
    if(head->prev==NULL){
        head=head_del(head);
        return head;
    }
    if(head->next==NULL){
        head=tail_del(head);
        return head;
    }

    head->prev->next=head->next;
    head->next->prev=head->prev;
    free(head);
    return;
```

```
    }
    if(head->next==NULL)
        return;
    removeEl(head->next,name_for_remove);
    return head;
}
```

Приложение Д.

```
int count(MusicalComposition* head){
    int i=0; //fdgfdgfdg
    MusicalComposition*cur=head;
    while(cur!=NULL){
        i++;
        cur=cur->next;
    }
    return i;
}
```

Приложение Е.

```
void print_names(MusicalComposition* head){
    if(head==NULL){ ///fdgdfgfdgfd
        return;
    }
    printf("Песня: %s \nИсполнитель %s\nГод: %i\n",head->name,head-
>author,head->year);
    print_names(head->next);
```

```
}
```

Приложение Ж.

```
MusicalComposition*  swap_list(MusicalComposition* head, char* a ,char* b){

   MusicalComposition* lst1=find(a,head); //dfgdfd
   MusicalComposition* lst2=find(b,head);

   MusicalComposition* p1=lst2->next;
   MusicalComposition* p2=lst1->prev;

   if(p2==NULL){
      lst2->next=lst1;
      lst2->prev=NULL;
      lst1->prev=lst2;
      lst1->next=p1;
      p1->prev=lst1;
      return lst2;
   }

   if(p1==NULL){
      lst1->next=NULL;
      lst1->prev=lst2;
      lst2->next=lst1;
```

```
        lst2->prev=p2;
        p2->next=lst2;
        return head;
    }

    lst1->next=lst2->next;
    lst1->prev=lst2;
    lst2->next=lst1;
    lst2->prev=p2;
    p2->next=lst2;
    p1->prev=lst1;
    return head;
}
```

Приложение 3.

```
MusicalComposition* sort(MusicalComposition* head){
    int k,i; //fdgfdg
    k=count(head);
    for(i=0;i<k;i++){
    if(head->next==NULL){
        return;
    }
    if(head->year > head->next->year){
        head = swap_list(head,head->name,head->next->name);
```

```
    }
    sort(head->next);
    }
    return head;
}
```

Приложение И.

```
MusicalComposition* head_del(MusicalComposition* head){ //dssdfdsfdsf
    MusicalComposition* newHead;
    if(head==NULL)
        return NULL;
    newHead= head->next;
    newHead->prev=NULL;
    free(head);
    return newHead;
}
```

Приложение К.

```
MusicalComposition* tail_del(MusicalComposition* head){
    if(head==NULL)
        return NULL;
    if(head->next==NULL){
        head->prev->next=NULL;
        free(head);
    }
```

```
    tail_del(head->next);
    return head;
}
```

Приложение Л.

```
MusicalComposition* find(char* a,MusicalComposition* head){
    int equal = 1;
    int i;
    if(head==NULL){
        return head;
    }
    while(head!=NULL){
        equal = 1;
        for(i=0;i<80;i++){
            if(head->name[i]=='\0')
                break;
            if(head->name[i]!=a[i])
                equal=0;
        }
            if(equal==1)
            return head;

            head=head->next;

    }
```

```
    return NULL;


}
```

Приложение M.

```
MusicalComposition* tail_add(MusicalComposition*
head,MusicalComposition* list){
    if(head->next!=NULL){//sadsa
        MusicalComposition* p=head->next;
        p->prev=list;
        list->next=p;
        return list;
    }
    head->next=NULL;
    head->prev=list;
    list->next=head;
    return list;


}
```

Приложение H.

```
MusicalComposition* add_elem(MusicalComposition*
head,MusicalComposition* list,char* a){
    MusicalComposition* list1=find(a,head);
     if(list1==NULL){
```

```
        printf("\n\nТакой музыкальной композиции не существует\n\n");
        return;
     }
    if(list1->next==NULL){
        head=push(head, list);
        return head;
    }
     MusicalComposition* p1=list1->next;



      list1->next=list;
      list->prev=list1;
      list->next=p1;
      p1->prev=list;
      return head;

}
```

Приложение О.

```
MusicalComposition* clean_list(MusicalComposition* head){
    if(head==NULL)
      return ;
    MusicalComposition* p;
    MusicalComposition* p1;
    while(head!=NULL){
```

```
        p=head;
        head=head->next;
    }
      p1=p;
    while(p1!=NULL){

        if(p1->prev!=NULL)
          p1->prev->next=NULL;

        p1->prev=NULL;
        p1=p1->prev;

        free(p1);
    }

    return NULL;

}
```

Приложение П.

```
int main(){

    int variable;
    int i=0,k=0;
    char name[80];
```

```
    char author[80];
    char name_for_remove[80];
    char name_add[80];
    int year;
    MusicalComposition* head=NULL;



    while(variable!=0){
    printf("Для выхода введите '0'\n");

    printf("Введите '1' для добавления элемента в список\nВведите '2', чтобы посмотреть список\nВведите '3', чтобы отсортировать список\n");
    printf("Введите '4' чтобы удалить элемент\nВведите '5' чтобы добавить элемент в начало списка\n");
    printf("Введите '6' чтобы добавить элемент после какого-либо элемента\nВведите '7' чтобы посчитать количество элементов списка\n");

    scanf("%i",&variable);


     switch ( variable ) {
        case 1:


            printf("Введите название музыкальной композиции:");
```

```
            getchar();
            fgets(name, 80, stdin);
            printf("Введите автора музыкальной композиции:");
            fgets(author, 80, stdin);
            printf("Введите год публикации музыкальной композиции:");
            scanf("%d", &year);
            getchar();
            (*strstr(name,"\n"))=0;
            (*strstr(author,"\n"))=0;
            if(i==0)
            head = createMusicalCompositionList(head,name, author, year);
            else {
              MusicalComposition* element_for_push =
createMusicalComposition(name, author, year);
              head=push(head,element_for_push);

            }
            i=1;
          break;

          case 2:
          printf("\n");
            print_names(head);
          printf("\n");
```

```
        break;


        case 3:
          head=sort(head);
          if(head!=NULL)
          printf("\n**********Список отсортирован**************\n");
          else
          printf("\n********** Заполните список :( **************\n");
        break;


        case 4:
        if(head!=NULL){
        printf("Введите название музыкальной композиции, чтобы удалить
её:");
        getchar();
        fgets(name_for_remove, 80, stdin);
        (*strstr(name_for_remove,"\n"))=0;
        head=removeEl(head, name_for_remove);
        }
        else
         printf("\nСписок пуст - нечего удалять :) \n\n");
        break;
        case 5:
```

```
            printf("Введите название музыкальной композиции:");
            getchar();
            fgets(name, 80, stdin);
            printf("Введите автора музыкальной композиции:");
            fgets(author, 80, stdin);
            printf("Введите год публикации музыкальной композиции:");
            scanf("%d", &year);
            getchar();
            (*strstr(name,"\n"))=0;
            (*strstr(author,"\n"))=0;
            if(i==0)
            head = createMusicalCompositionList(head,name, author, year);
            else {
              MusicalComposition* element_for_push =
createMusicalComposition(name, author, year);
              head=tail_add(head,element_for_push);

            }
            i=1;
          break;
          case 6:
            if(head!=NULL){
```

```
printf("Введите название музыкальной композиции, после которой хотите добавить элемент:");
getchar();
fgets(name, 80, stdin);
printf("Введите название музыкальной композиции, которую хотите добавить:");
fgets(name_add, 80, stdin);
printf("Введите автора музыкальной композиции, которую хотите добавить:");
fgets(author, 80, stdin);
printf("Введите год публикации музыкальной композиции, которую хотите добавить:");
scanf("%d", &year);
getchar();
(*strstr(name,"\n"))=0;
(*strstr(name_add,"\n"))=0;
(*strstr(author,"\n"))=0;
MusicalComposition* element_for_push = createMusicalComposition(name_add, author, year);


  head=add_elem(head,element_for_push,name);


  }
  else
```

```
            printf("Список пуст\n");

        break;
        case 7:

        printf("В списке: %d(-ов)\n\n",count(head));
        break;
        case 8:

        break;
}
    }
    head=clean_list(head);
    free(head);


    return 0;

}
```

Приложение Р.

```
all: main.o
        gcc main.o
main.o: main.c
        gcc -c main.c
```

```
clean:
	rm main.o
```

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1) Язык программирования СИ / Керниган Б., Ритчи Д. СПб.: Издательство "Невский Диалект", 2001. 352 с
2) UNIX. Программное окружение / Керниган Б., Пайк Р. СПб.: Символ Плюс, 2003. 416 с.