**Министерство образования науки Российской Федерации**

Федеральное государственное бюджетное образовательное учреждение высшего образования «Московский авиационный институт (национальный исследовательский университет)»

«Информационные технологии и прикладная математика»

**Курсовой проект**

**По курсу «Практикум на ЭВМ»**

**2 семестр**

**Задание 7:**

«Линейные списки»

|  |  |
| --- | --- |
| Группа: | М8О-106Б-21 |
| Студент: | Орусский В.Р. |
| Преподаватель: | Дубинин А.В. |
| Оценка: |  |
| Дата: |  |

**Оглавление**

[Введение 3](#_Toc104750260)

[Постановка задачи 4](#_Toc104750261)

[Вариант 5-3-11: 4](#_Toc104750262)

[Теоретическая часть 4](#_Toc104750263)

[Описание алгоритма 4](#_Toc104750264)

[Использованные в программе переменные 5](#_Toc104750265)

[Пользовательские структуры данных в программе: 5](#_Toc104750266)

[Исходники программы 6](#_Toc104750267)

[Протокол исполнения и тесты 10](#_Toc104750268)

[Вывод 13](#_Toc104750269)

# Введение

Изучить функционал и реализацию линейного списка (связный список), где берётся первый элемент, а к нему последовательно подвязываются все последующие.

Написать линейный список для заданного типа данных с стандартным функционалом и дополнить его реализацией нестандартного действия.

# Постановка задачи

Составить и отладить программу на языке Си для обработки линейного списка заданной организации с отображением списка на динамические структуры. Навигацию по списку следует реализовать с применением итераторов. Предусмотреть выполнение одного нестандартного действия и четырёх стандартных действий.

Стандартные действия:

1. Печать списка;

2. Вставка/Удаление элемента в список;

3. Подсчёт длины списка.

## Вариант 5-3-11:

Реализовать линейный однонаправленный с барьерным элементом список на итераторах, с литерным типом данных. Нестандартное действие – переворот списка наоборот (reverse).

# Теоретическая часть

Связный (линейный) список – динамическая структура данных, в которой каждый из элементов содержит, как собственное значение, так и одну или две ссылки на след или предыдущий элемент списка. Именно по кол-ву этих ссылок списки делятся на односвязные и двусвязные списки.

Преимуществом данной структуры является её гибкость. Данные в памяти могут располагаться не последовательно, даже в абсолютно разных местах памяти, но порядок обхода будет работать корректно благодаря внутренним связкам.

Барьерный элемент – элемент, указывающий на конец списка.

# Описание алгоритма

В начале были реализованы итераторы. Это фундамент для любой структуры данных, где есть связки между элементами. Они работают максимально просто, содержат в себе ссылку на элемент списка (который в свою очередь включает в себя значение элемента, ссылку на след элемент).

Для итераторов были реализованы функции создания, возвращения след итератора, получать узел и устанавливать новый.

После уже был реализован сам список, в которым были стандартные функции, которые указаны выше, а также создание, полное удаление и нестандартное действие «ListReverse».

На нём остановимся подробнее, поскольку функционал остальных функций понятен из названия и реализация относительно простая.

У реверса списка есть пара нюансов, так как просто переставить элементы, не потеряв связи очень проблематично, а терять их не хочется…

Первый элемент меняется с последним, второй с предпоследним. Параллельно запоминаются пред./след. от левого и пред. от правого элементы.

# Использованные в программе переменные

|  |  |  |
| --- | --- | --- |
| Название | Тип данных | За что отвечает |
| rubish | char | Просто заглушка, дабы корректно работал текстовый интерфейс |
| c | char | Взаимодействие с пользователем |
| data | char | Значение для элемента в списке |
| lst | list | Основной список |
| it | listIterator | Итератор по основному списку |
| tail | list\_node | Барьер |

# Пользовательские структуры данных в программе:

|  |  |  |
| --- | --- | --- |
| Название | Поля | Значения |
| listIterator | list\_node node | Указатель на элемент списка |
| list\_node | item data | Собственное значение элемента списка |
| list\_node next | Ссылка на следующий элемент списка (связка) |
| list | list\_node head | Ссылка на главный (самый первый) элемент списка |

# Исходники программы

Файлы «iterator.h» и «list.h» – заголовочные файлы, в котором объявлена структура итератора и списка

#ifndef **ITERATOR\_H**#define **ITERATOR\_H**#include "list.h"  
  
*typedef struct* \_listIterator {  
 list\_node \*node;  
} listIterator;  
  
listIterator \*IteratorCreate(list \*lst);  
*void* IteratorNext(listIterator \*it);  
list\_node \*IteratorGet(listIterator \*it);  
*void* IteratorSet(list\_node \*l, listIterator \*it);  
  
#endif

#ifndef **LIST\_H**#define **LIST\_H***typedef char* item;  
*typedef struct* \_list\_node list\_node;  
  
*// Структура, отвечающая за элемент списка  
typedef struct* \_list\_node {  
 item data;  
 list\_node \*next;  
} list\_node;  
  
*// Сам список, в котором есть указатель на голову (главный элемент)  
typedef struct* {  
 list\_node \*head;  
} list;  
  
list \*ListCreate();  
*void* ListPrint(list \*l);  
*void* ListInsert(list \*lst, item data);  
*void* ListRemove(list \*lst, item data);  
*int* ListLen(list \*lst);  
*void* ListReverse(list\_node \*lstFromHead, list\_node \*lstFromTail, *int* i, list \*lst, *int* len);  
  
#endif

Файлы «iterator.c» и «list.c» - реализация указанных выше функций.

#include <stdlib.h>  
#include <stdio.h>  
#include "iterator.h"  
  
*// Создаём итератор от головы списка*listIterator \*IteratorCreate(list \*lst) {  
 *if* (lst != **NULL**) {  
 listIterator \*it = (listIterator \*)malloc(*sizeof*(listIterator));  
 it->node = lst->head;  
 *return* it;  
 } *else return* **NULL**;  
}  
  
*// Переводим итератор на след элемент  
void* IteratorNext(listIterator \*it) {  
 *if* (it->node == **NULL**) {  
 printf("You have met a barrier element! The list is finished\n");  
 } *else* {  
 it->node = it->node->next;  
 }  
}  
  
*// Возвращаем итератор*list\_node \*IteratorGet(listIterator \*it) {  
 *return* it->node;  
}  
  
*// Устанавливаем новую связь для итератора  
void* IteratorSet(list\_node \*l, listIterator \*it) {  
 it->node = l;  
}

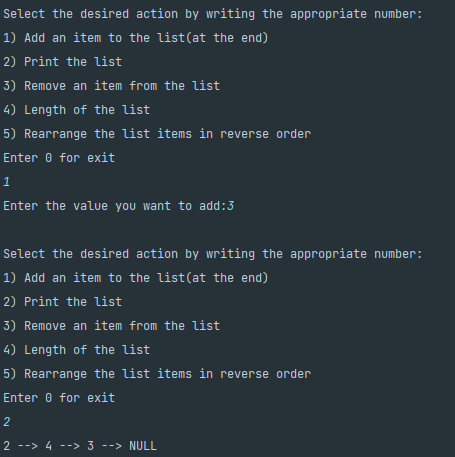
#include <stdio.h>  
#include <stdlib.h>  
#include "list.h"  
#include "iterator.h"  
  
*// Создание списка*list \*ListCreate() {  
 list \* lst = (list \*)malloc(*sizeof*(list));  
 lst->head = (list\_node \*)malloc(*sizeof*(list\_node));  
 lst->head->next = **NULL**;  
 lst->head->data = ' ';  
 *return* lst;  
}  
  
*// Вывод списка, барьер в виде NULL  
void* ListPrint(list \*l) {  
 listIterator \*it = IteratorCreate(l);  
 *if* (it->node) {  
 *while* (it->node != **NULL**) {  
 *if* (it->node->data != ' ') printf("%c --> ", it->node->data);  
 it->node = it->node->next;  
 }  
 printf("NULL");  
 printf("\n");  
 }  
}  
  
*// Добавление элемента в список  
void* ListInsert(list \*lst, item data) {  
 listIterator \*it = IteratorCreate(lst);  
 *if* (it->node) {  
 *while* (it->node->next) *// next item is NULL!* {  
 IteratorNext(it);  
 }  
 list\_node \*tail = (list\_node \*)malloc(*sizeof*(list\_node));  
 tail->next = **NULL**;  
 tail->data = ' ';  
 it->node->data = data;  
 it->node->next = tail;  
 } *else* printf("The list doesn't exist!\n");  
 free(it);  
}  
  
*// Удаление всего списка  
void* ListRemove(list \*lst, item data) {  
 *int* flag = 0;  
 listIterator \*it = IteratorCreate(lst);  
 list\_node \*prew;  
 *if* (it->node) {  
 *if* (it->node->data == data) { *// case, then out node is head* lst->head = it->node->next;  
 free(it->node);  
 it->node = **NULL**;  
 } *else* {  
 *while* (it->node->next) {  
 *if* (it->node->next->data == data) *// case, then our node between two elements or is tail* {  
 prew = it->node;  
 IteratorNext(it);  
 flag = 1;  
 *if* (it->node->next != **NULL**) {  
 prew->next = it->node->next;  
 }  
 free(it->node);  
 it->node = **NULL**;  
 *break*;  
 }  
 IteratorNext(it);  
 }  
 *if* (!flag) printf("There is no such element\n");  
 }  
 } *else* printf("The list doesn't exist!\n");  
 free(it);  
}  
  
*// Длина списка  
int* ListLen(list \*lst) {  
 *int* count = 0;  
 listIterator \*it = IteratorCreate(lst);  
 IteratorSet(lst->head, it);  
 *if* (it->node) {  
 *while* (it->node->next && it->node->next->data != ' ') {  
 count++;  
 IteratorNext(it);  
 }  
 } *else* printf("The list doesn't exist!\n");  
 free(it);  
 count++;  
 *return* count;  
}  
  
*// Переворот списка (нестандартное действие)  
void* ListReverse(list\_node \*lstFromHead, list\_node \*lstFromTail, *int* i, list \*lst, *int* len) {  
 *if* (i - 1 == len / 2) *return*;  
 list\_node \*lstFromHeadNext = lstFromHead->next;  
 list\_node \*lstFromTailPrew; *// prew from lstFromTail* list\_node \*lstFromHeadPrew; *// prew from lstFromHead* listIterator \*it = IteratorCreate(lst);  
 IteratorSet(lstFromHead, it);  
 *for* (*int* j = 0; j < len - 2 \* i; j++) {  
 IteratorNext(it);  
 }  
 lstFromTailPrew = it->node;  
 IteratorSet(lst->head, it);  
 *for* (*int* j = 0; j < i - 2; j++) {  
 IteratorNext(it);  
 }  
 lstFromHeadPrew = it->node;  
 *if* (i == 1) {  
 lst->head = lstFromTail;  
 lstFromTail->next = lstFromHeadNext;  
 lstFromHead->next = **NULL**;  
 lstFromTailPrew->next = lstFromHead;  
 } *else* {  
 lstFromTailPrew->next = lstFromHead;  
 lstFromHead->next = lstFromTail->next;  
 *if* (2 \* i != len) {  
 lstFromTail->next = lstFromHeadNext;  
 } *else* {  
 lstFromTail->next = lstFromHead;  
 }  
 lstFromHeadPrew->next = lstFromTail;  
 }  
 i++;  
 free(it);  
 ListReverse(lstFromHeadNext, lstFromTailPrew, i, lst, len);  
}

Файл «main.c» - файл для взаимодействия с пользователем и реализация функций, выполняющих основное поставленное задание.

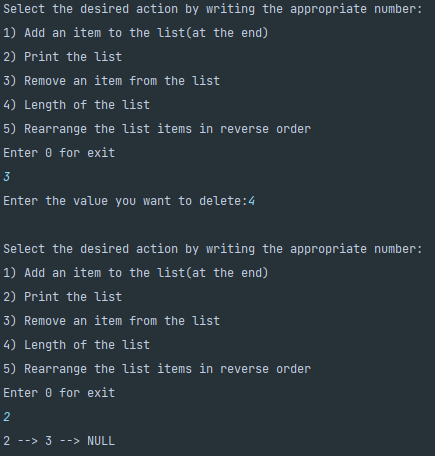
#include <stdio.h>  
#include <stdlib.h>  
#include "list.h"  
#include "iterator.h"  
  
*void* print\_menu() {  
 printf("\nSelect the desired action by writing the appropriate number:\n");  
 printf("1) Add an item to the list(at the end)\n");  
 printf("2) Print the list\n");  
 printf("3) Remove an item from the list\n");  
 printf("4) Length of the list\n");  
 printf("5) Rearrange the list items in reverse order\n");  
 printf("Enter 0 for exit\n");  
}  
  
*int* main(*void*) {  
 item data;  
 *char* rubish;  
 list \*lst = ListCreate();  
 listIterator \*it = IteratorCreate(lst);  
 *char* c;  
 printf("Welcome to the list processing program!\n");  
 print\_menu();  
 *while* ((c = getchar()) != **EOF**) {  
 IteratorSet(lst->head, it);  
 *if* (c == '\n' || c == ' ') *continue*;  
 *switch* (c) {  
 *case* '1': printf("Enter the value you want to add: ");  
 rubish = getchar();  
 data = getchar();  
 ListInsert(lst, data);  
 *break*;  
 *case* '2': ListPrint(lst);  
 *break*;  
 *case* '3': printf("Enter the value you want to delete: ");  
 rubish = getchar();  
 data = getchar();  
 ListRemove(lst, data);  
 *break*;  
 *case* '4': printf("%d", ListLen(lst));  
 *break*;  
 *case* '5': IteratorSet(lst->head, it);  
 *for* (*int* j = 0; j < ListLen(lst) - 1; j++) { *// search for tail list\_node* IteratorNext(it);  
 }  
 ListReverse(lst->head, it->node, 1, lst, ListLen(lst));  
 IteratorSet(lst->head, it);  
 *for* (*int* j = 0; j < ListLen(lst) - 1; j++) { *// search for tail list\_node* IteratorNext(it);  
 }  
 list\_node \*tail = (list\_node \*)malloc(*sizeof*(list\_node));  
 tail->next = **NULL**;  
 tail->data = ' ';  
 it->node->next = tail;  
 *break*;  
 *case* '0': printf("\nHave a nice day!\n");  
 free(it);  
 free(lst);  
 *return* 0;  
 *default*: printf("The number you entered isn't in the menu :(\nPlease try again, in case you made a mistake\n");  
 *break*;  
 }  
 print\_menu();  
 }  
}

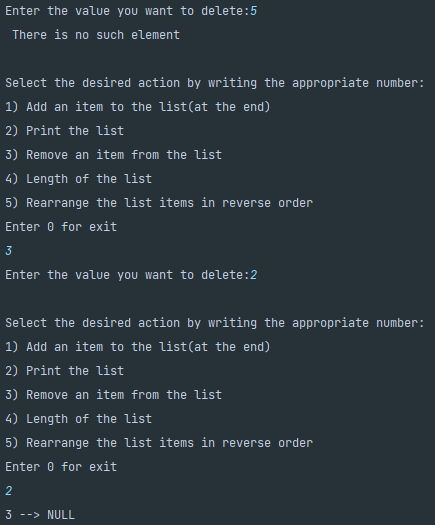
# Протокол исполнения и тесты

Тест №1

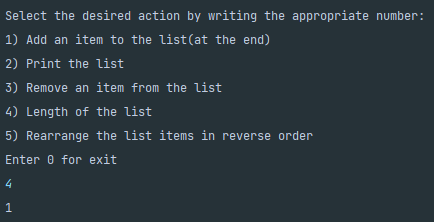


Тест №2

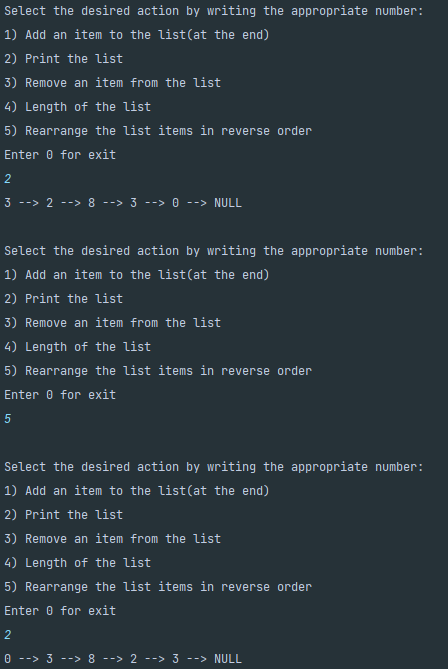


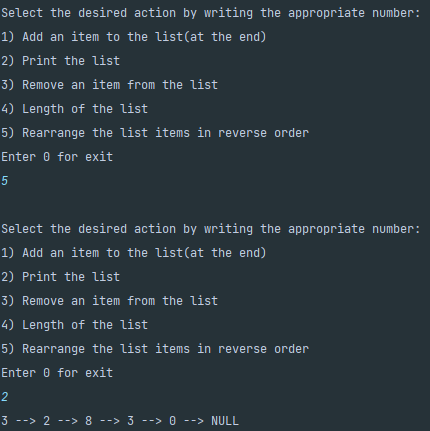


Тест №3



Тест №4 (дважды перевернули)





# Вывод

Данная работа очень полезна для будущего развития в сфере IT. Поскольку, по моему мнению, да и в целом, списки – фундамент для огромного кол-ва других структур данных, которые незаменимы в современном production мире. Более того, благодаря этой работе, я смог более подробно изучить принцип работы итераторов, а также реализовать их.