Министерство образования Республики Беларусь

ПОЛОЦКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Кафедра технологий программирования

**Методические указания  
 к лабораторной работе № 2\_7  
 по курсу «Основы алгоритмизации   
и программирования»**

«Динамические структуры данных.

Списки»

Преподаватель: Войтехович   
Агния Витольдовна

Составитель: Войтехович   
Агния Витольдовна

Полоцк, 2017

# **ЦЕЛЬ РАБОТЫ**

Изучить методы организации списочных структур в динамической памяти. Реализовать алгоритмы добавления, удаления и сортировки списков.

ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

# 1 Введение в абстрактные структуры

Абстрактные структуры данных предназначены для удобного хранения и доступа к информации. Они предоставляют удобный интерфейс для типичных операций с хранимыми объектами, скрывая детали реализации от пользователя. Конечно, это весьма удобно и позволяет добиться большей модульности программы. Абстрактные структуры данных иногда делят на две части: интерфейс, набор операций над объектами, который называют АТД (абстрактный тип данных) и реализацию.

Каждый уже имел дело с простыми абстрактными структурами данных - например, когда оперировал с числами. Языки программирования высокого уровня (Паскаль, Си..) предоставляют удобный интерфейс для чисел: операции +, \*, = .. и т.п, но при этом скрывают саму реализацию этих операций, машинные команды.

# 2 Статические структуры данных

Статические структуры относятся к разряду непримитивных структур, которые, фактически, представляют собой структурированное множество примитивных, базовых, структур. Например, вектор может быть представлен упорядоченным множеством чисел. Поскольку по определению статические структуры отличаются отсутствием изменчивости, память для них выделяется один раз и ее объем остается неизменным до уничтожения структуры. Слово 'статический' относится скорее к реализации структуры, нежели к АТД.

Простейшая статическая структура данных - массив, где обращение к элементу происходит через его номер.

Слово 'массив' употребляется в различных контекстах:

* как АТД, т.е множество с операциями:
  + получить элемент с номером N;
  + записать элемент с номером N;
* и как физическая структура, реализованная в виде непрерывной области памяти.

В случае реализации массива через такую структуру, номер соответствует смещению от начала области.

**Плюсов у массива**(здесь реализация) всего два, но зато больших:

* доступ за константное время к любому элементу
* память тратится только на данные\*

\* - здесь и далее будут опущены константные затраты памяти операционной системой, возникающие при реализации структуры. В частности, в большинстве ОС, при динамическом выделении памяти под массив, в начале соответствующей области памяти ставится специальная метка.

**Минус** - один, но тоже большой: статичность, неизменность структуры. Одномерный массив иногда называют ***вектором***.

# 3 Динамические структуры данных

Динамические структуры по определению характеризуются отсутствием физической смежности элементов структуры в памяти непостоянством и непредсказуемостью размера (числа элементов) структуры в процессе ее обработки.

Поскольку элементы динамической структуры располагаются по непредсказуемым адресам памяти, адрес элемента такой структуры не может быть вычислен из адреса начального или предыдущего элемента. Для установления связи между элементами динамической структуры используются указатели, через которые устанавливаются явные связи между элементами. Такое представление данных в памяти называется связным. Элемент динамической структуры состоит из двух полей:

1) информационного поля или поля данных, в котором содержатся те данные, ради которых и создается структура; в общем случае информационное поле само является интегрированной структурой - вектором, массивом, другой динамической структурой и т.п.;

2) поле связок, в котором содержатся один или несколько указателей, связывающий данный элемент с другими элементами структуры;

Когда связное представление данных используется для решения прикладной задачи, для конечного пользователя "видимым" делается только содержимое информационного поля, а поле связок используется только программистом-разработчиком.

**Достоинства связного представления данных** - в возможности обеспечения значительной изменчивости структур;

* размер структуры ограничивается только доступным объемом машинной памяти;
* при изменении логической последовательности элементов структуры требуется не перемещение данных в памяти, а только коррекция указателей;
* большая гибкость структуры.

Вместе с тем связное представление не лишено и **недостатков**, основные из которых:

* на поля связок расходуется дополнительная память;
* доступ к элементам связной структуры может быть менее эффективным по времени.

Последний недостаток является наиболее серьезным и именно им ограничивается применимость связного представления данных. Если в смежном представлении данных для вычисления адреса любого элемента нам во всех случаях достаточно было номера элемента и информации, содержащейся в дескрипторе структуры, то для связного представления адрес элемента не может быть вычислен из исходных данных. Дескриптор связной структуры содержит один или несколько указателей, позволяющих войти в структуру, далее поиск требуемого элемента выполняется следованием по цепочке указателей от элемента к элементу. Поэтому связное представление практически никогда не применяется в задачах, где логическая структура данных имеет вид вектора или массива - с доступом по номеру элемента, но часто применяется в задачах, где логическая структура требует другой исходной информации доступа (таблицы, списки, деревья и т.д.).

# 4 CПИСКИ

***Списком*** называется упорядоченное множество, состоящее из переменного числа элементов, к которым применимы операции включения, исключения.

Список, отражающий отношения соседства между элементами, называется ***линейным***. Длина списка равна числу элементов, содержащихся в списке, список нулевой длины называется ***пустым списком***. Линейные связные списки являются простейшими динамическими структурами данных.

Графически связи в списках удобно изображать с помощью стрелок. Если компонента не связана ни с какой другой, то в поле указателя записывают значение, не указывающее ни на какой элемент. Такая ссылка обозначается специальным именем - nil.

На рисунке 1 приведена структура односвязного списка. На нем поле INF - информационное поле, данные, NEXT - указатель на следующий элемент списка. Каждый список должен иметь особый элемент, называемый указателем начала списка или головой списка, который обычно по формату отличен от остальных элементов. В поле указателя последнего элемента списка находится специальный признак nil, свидетельствующий о конце списка.

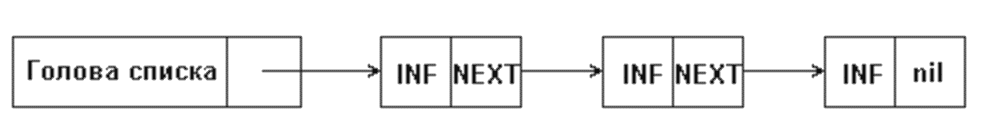


Рисунок 1 - Представление односвязного списка в памяти

***Двусвязный список*** характеризуется наличием пары указателей в каждом элементе: на предыдущий элемент и на следующий:

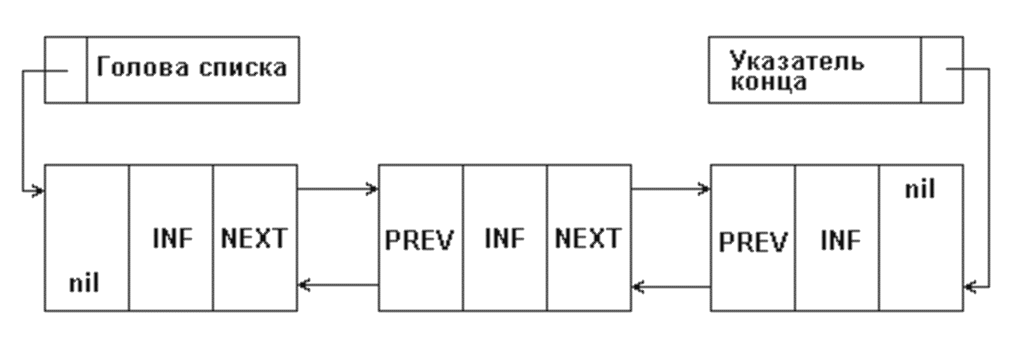


Рисунок 2 - Представление двусвязного списка в памяти

Очевидный плюс тут в том, что от данного элемента структуры мы можем пойти в обе стороны. Таким образом упрощаются многие операции. Однако на указатели тратится дополнительная память.

Разновидностью рассмотренных видов линейных списков является ***кольцевой список***, который может быть организован на основе как односвязного, так и двухсвязного списков. При этом в односвязном списке указатель последнего элемента должен указывать на первый элемент; в двухсвязном списке в первом и последнем элементах соответствующие указатели переопределяются, как показано на рисунке 3.

При работе с такими списками несколько упрощаются некоторые процедуры. Однако, при просмотре такого списка следует принять некоторые меры предосторожности, чтобы не попасть в бесконечный цикл.

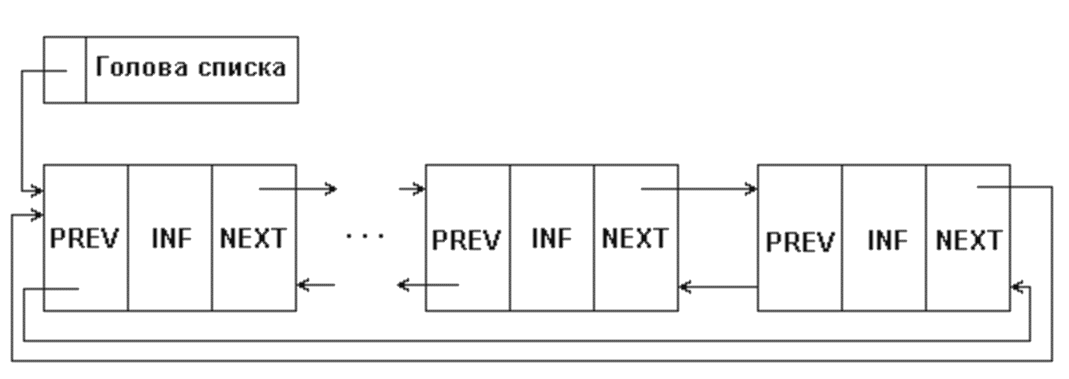


Рисунок 3 - Структура кольцевого двухсвязного списка

Описываемые ниже АТД могут быть организованы на базе

* массива: выделяется место под N элементов разом, а затем описываются операции над данным типом данных в терминах операций над элементами массива.
* списка: память выделяется и освобождается по мере необходимости.

Первый вариант быстрее, но ***лишь второй истинно динамический***. Соответственно, в различных приложениях может быть предпочтителен первый (размер структуры известен и небольшой) или второй (размер заранее неизвестен). Мы будем рассматривать преимущественно динамические решения.

## 4.1 Линейный список

Чтобы Вам стало более понятно, что такое линейный список, рассмотрим рисунок 1.

Каждый прямоугольник на этом рисунке — элемент списка. Как и было сказано выше, в каждом элементе есть поле с данными и поле с указателем на следующий элемент. Следует отметить, значение **nil** у последнего прямоугольника — признак конца списка.

Вообще говоря, чаще всего списки используются с фиктивным элементом, который является первым в последовательности. Использование фиктивного элемента позволяет упростить реализацию некоторых операций. Добавлю, что не стоит пугаться таких элементов, это всего-навсего элемент списка, поле данных которого не используется.

Над линейный списком определены следующие операции:

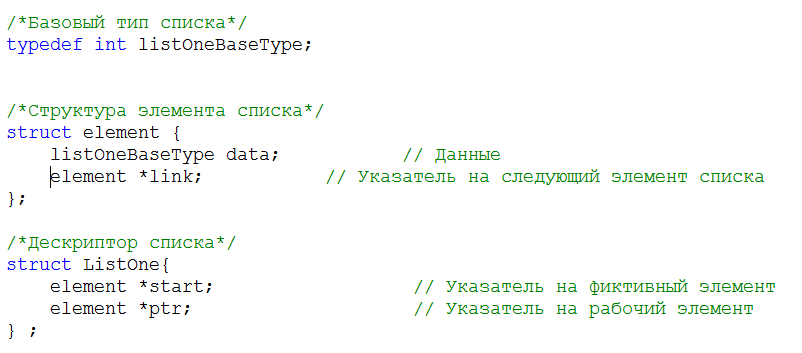
1. инициализация;
2. включение элемента;
3. исключение элемента;
4. переход в начало списка;
5. переход в конец списка;
6. переход к следующему элементу;
7. уничтожение списка.

## 4.1.1 Реализация односвязного линейного списка (ОЛС)

При реализации ОЛС в статической памяти располагают дескриптор, состоящий из двух полей:

1. указатель на фиктивный элемент;

2. указатель на текущий элемент.



Хочу обратить ваше внимание на тот факт, что в любой момент времени нам известен лишь дескриптор и, соответственно, информация в нем, поэтому все операции над списком выполняются непосредственно через него.

**ВНИМАНИЕ!** При использовании фиктивного элемента, все операции нужно выполнять после того элемента, на который указывает ptr.

## 4.1.2 Инициализация списка

Инициализация происходит в три шага:

1. Выделение памяти для фиктивного элемента.

2. В дескрипторе указателю start присвоить блок с только что выделенной памятью.

3. Указателю ptr присвоить start.

Поясню, что здесь, и зачем. При инициализации происходит создание списка, но не заполнение его элементами. Создание списка подразумевает создание фиктивного элемента. Так как его поле данных не используется, то список остается по-прежнему пуст. Когда память выделена, остается лишь правильно присвоить ссылки указателям (рисунок 4).

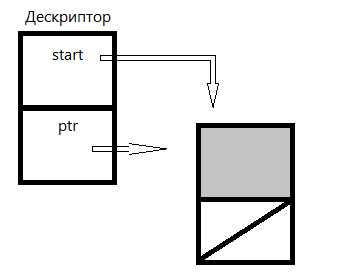
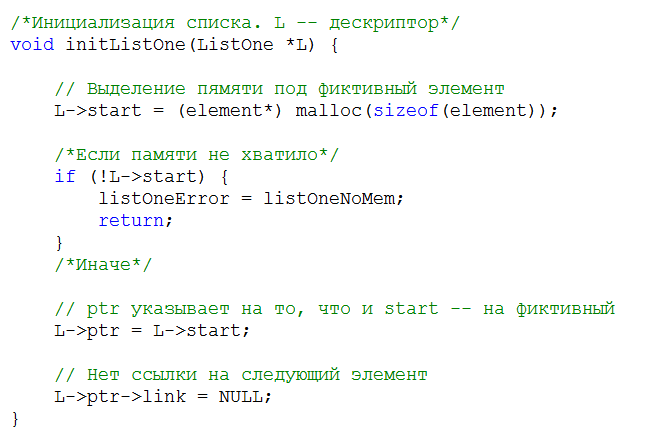


Рисунок 4 – Инициализация нового списка

Код инициализации таков:



## 4.1.3 Включение элемента

Включение элемента порой сложно понять, поэтому этот вопрос будет рассмотрен максимально подробно. Рассмотрим список из нескольких элементов, изображенный на рисунке 5.

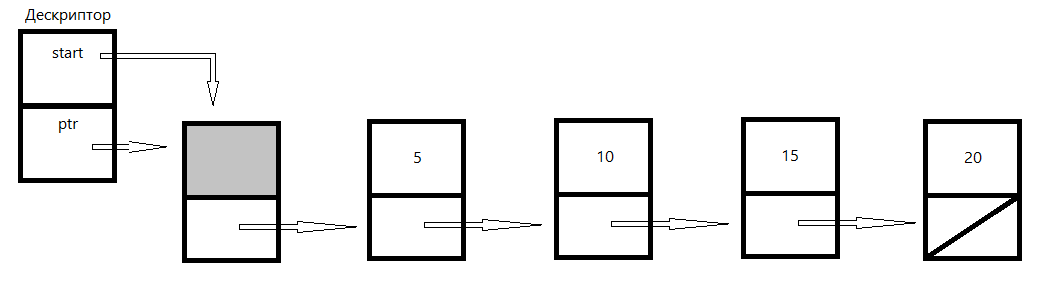


Рисунок 5 – Исходный список

Так как ptr указывает на фиктивный элемент, то новый элемент нужно вставить после него, но перед элементом со значением 5. Прежде чем вставить, нужно выделить память под новый элемент и занести информацию в него. Допустим, мы вставляем элемент с данными — 100 (рисунок 6).

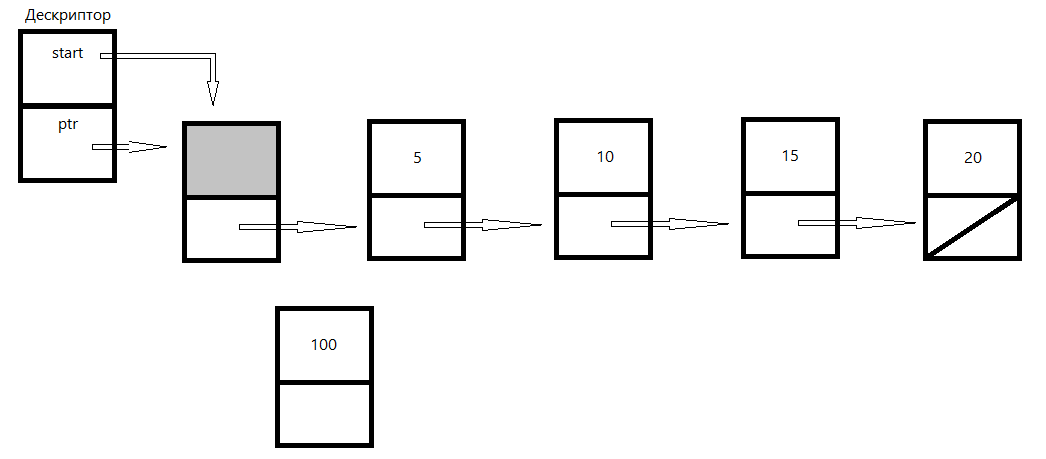


Рисунок 6 – Исходный список и новый элемент,  
 не связанные друг с другом

Теперь нужно вставить новый элемент между двумя элементами списка, то есть изменить ссылки, находящиеся в полях ptr, как это показано на рисунке 7.

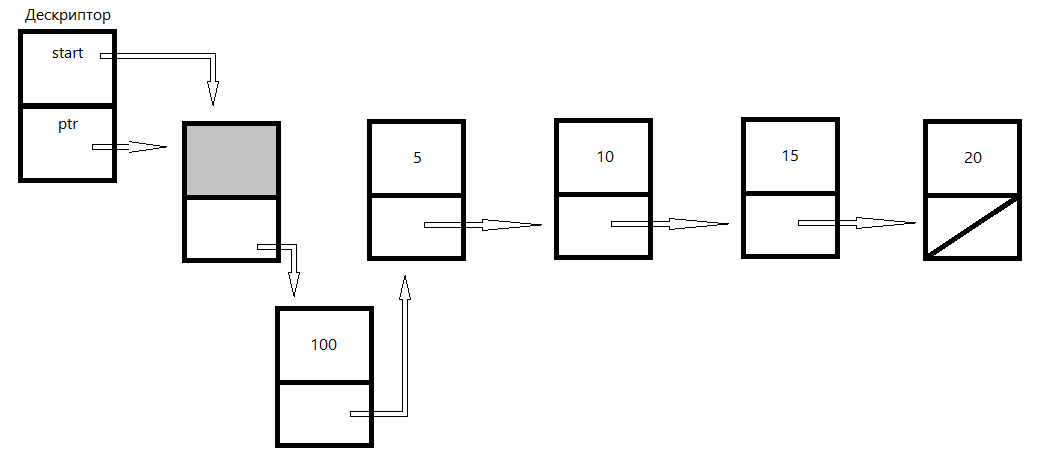
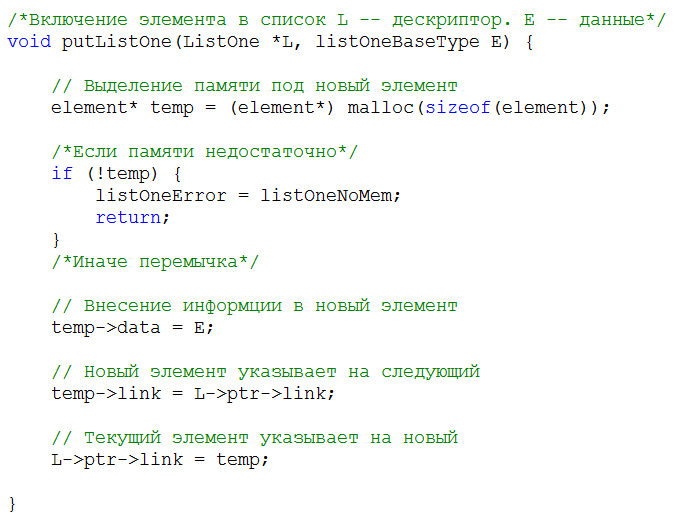


Рисунок 7 – Список после включения элемента

Теперь рассмотрим код операции включения:



## 4.1.4 Исключение элемента

Пусть нам дан тот же список, что и в примере с включением элемента (рисунок 5).

Так как ptr указывает на фиктивный — исключаем первый. Здесь перемычка ссылок немного сложнее, чем в предыдущем примере. Нужно создать временный элемент так, чтобы он был равен исключаемому, т.е. имел один и тот же с ним адрес. *Это нужно для того, чтобы не потерять тот элемент, на который исключаемый ссылается*. Данный процесс изображен на рисунке 8

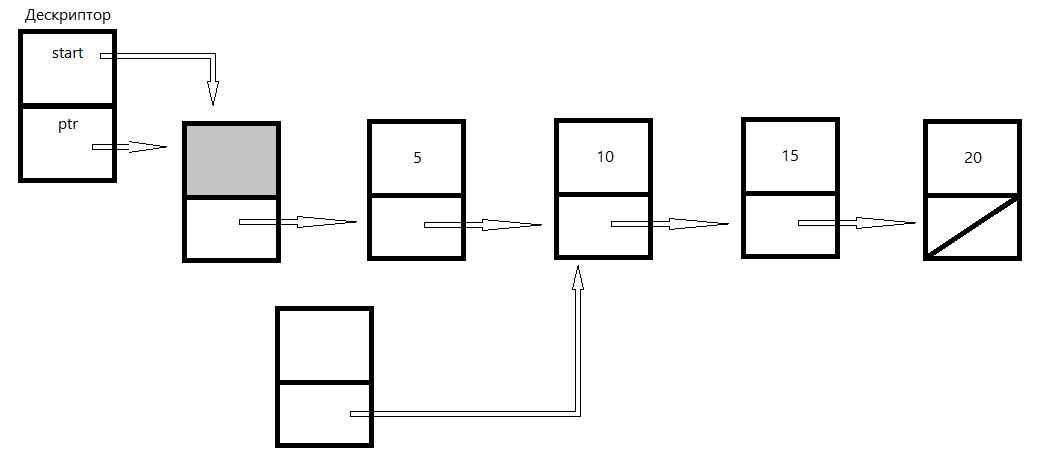


Рисунок 8 – Подключение к списку временного элемента

Теперь необходимо произвести перемычку указателей и удаление временного элемента, как показано на рисунке 9.

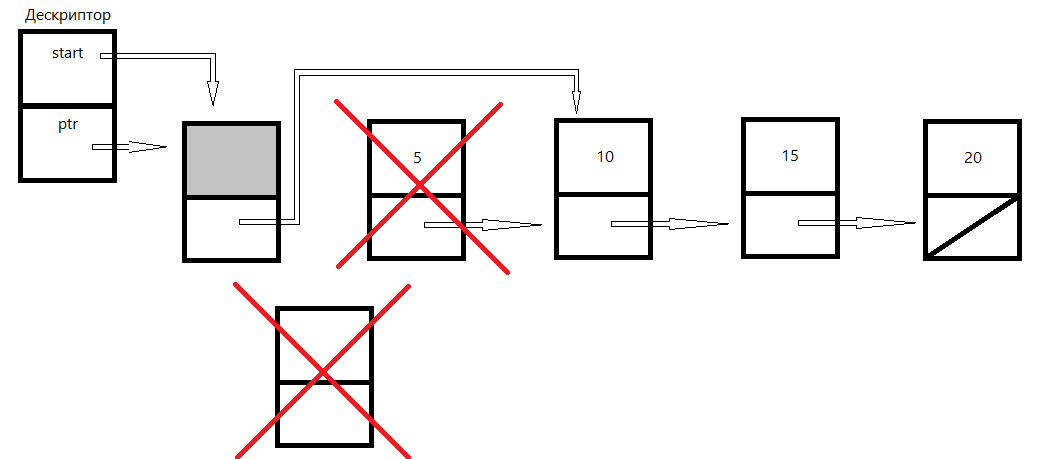
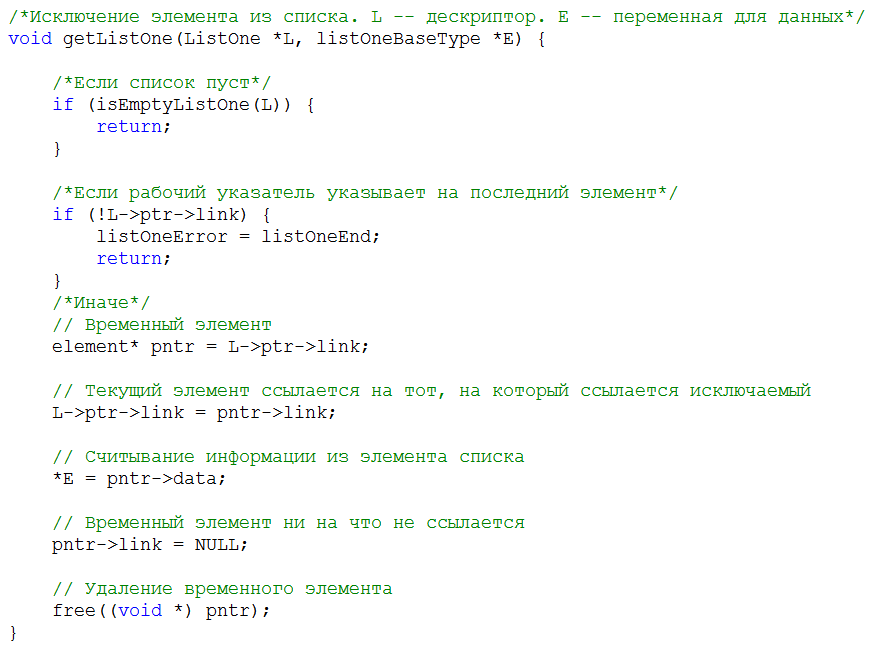


Рисунок 9 – Перемычка указателей и удаление лишних элементов

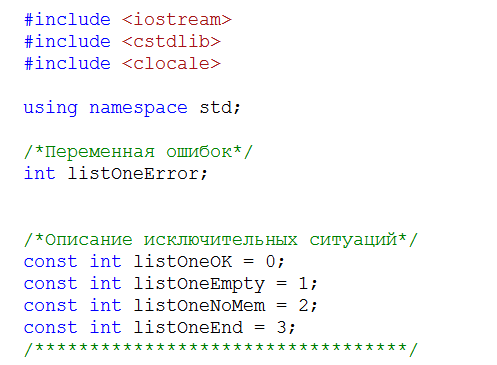
Вот и вся операция исключения. Ниже представлен код:



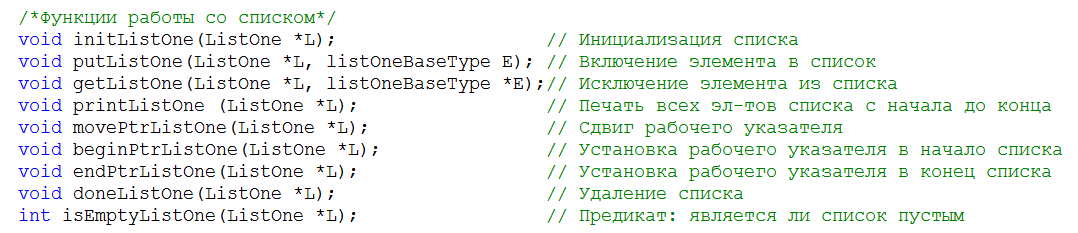
Остальные операции нет смысла так подробно расписывать, так как они просты в реализации.

## 4.1.5 Реализация остальных функций линейного списка

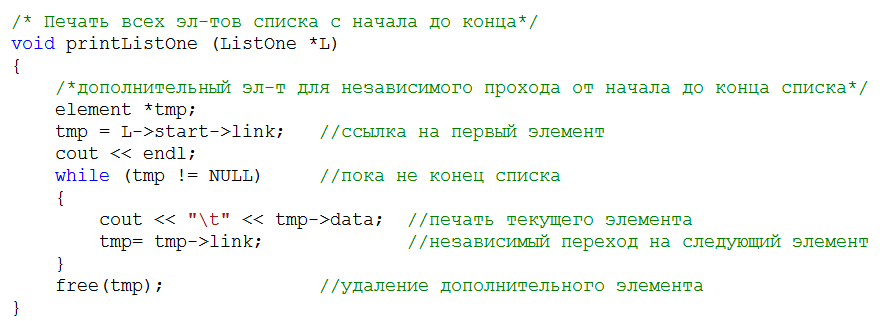
Для начала, подключим к проекту необходимые библиотеки и объявим глобальные переменные, которые помогут организовать корректную работу итогового приложения.

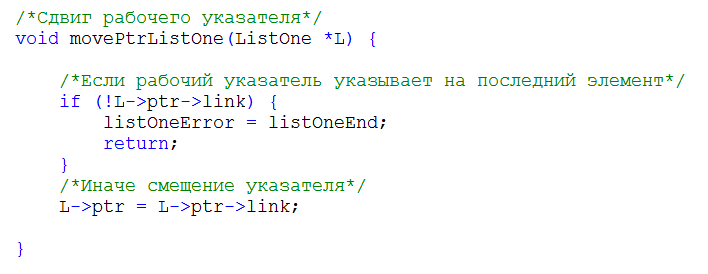


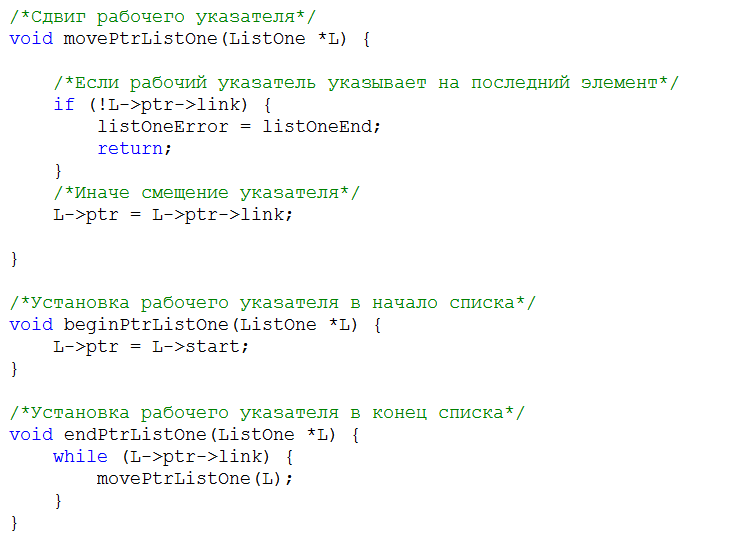
Нам необходимо реализовать следующие пользовательские функции для работы со списком:

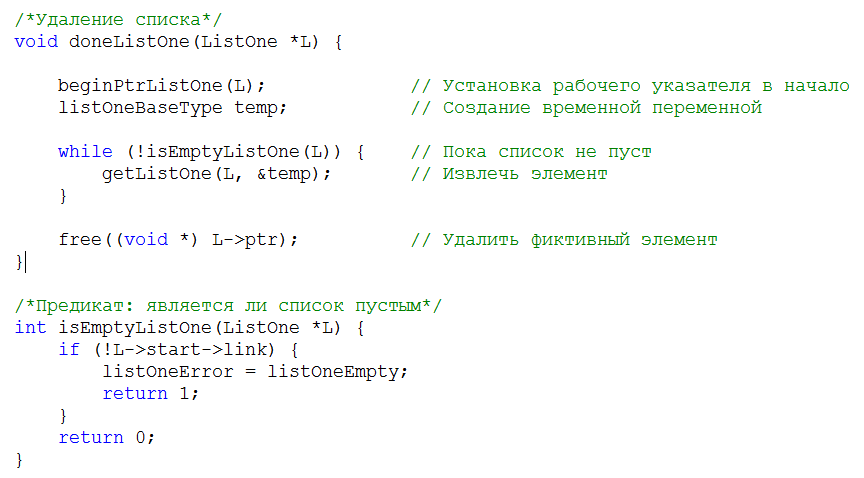


Функции ***initListOne()***, ***putListOne()*** и ***getListOne()*** были описаны выше, поэтому их листинг приводить повторно не будем, а вот реализация всех остальных функций представлена ниже.



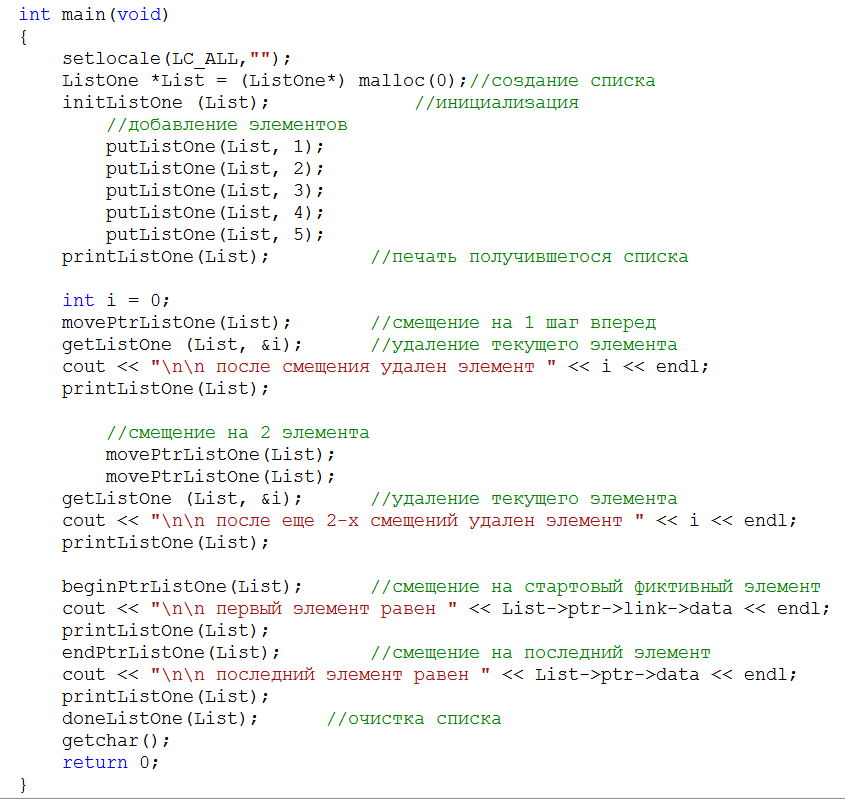






## 4.1.6 Тестирование функций линейного списка

Проверим работу описанных выше функций, написав следующее содержимое функции ***main():***



На рисунке 10 изображен результат тестирования программы.

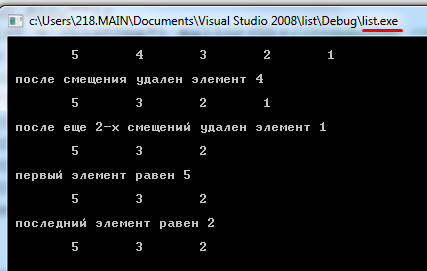


Рисунок 10 – Результат тестирования функций   
работы с линейным списком

Осталось только добавить, что перечисленные выше операции — далеко не все, которые можно реализовать для списка. Вы сами можете создавать новые, исходя из поставленных перед вами задач.

## 4.2 Двусвязный список

Так как каждый элемент двусвязного линейного списка (ДЛС) содержит, помимо поля с данными, два указателя, то отсюда вытекает то, что для ДЛС требуется больше памяти.

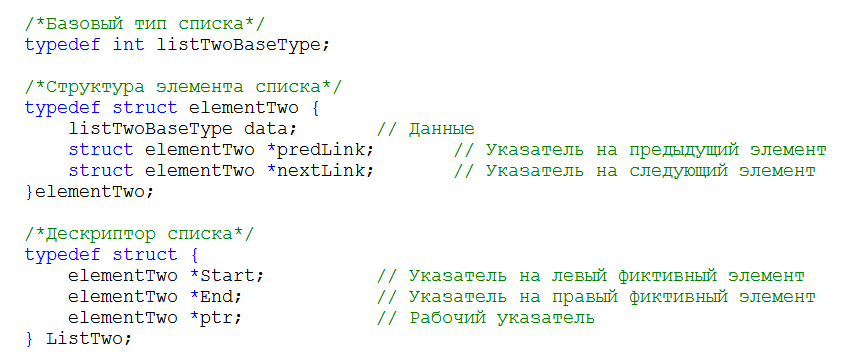
Использование двух указателей дает несколько преимуществ. Самым главным из которых является то, что перемещаться по списку можно в двух направлениях, благодаря чему упрощаются некоторые операции (по сравнению с ОЛС) и добавляются новые.

**Операции над ДЛС:**

1. инициализация;
2. включение до рабочего указателя;
3. включение после рабочего указателя;
4. исключение до рабочего указателя;
5. исключение после рабочего указателя;
6. сдвиг рабочего указателя назад (к предыдущему элементу);
7. сдвиг рабочего указателя вперед (к следующему элементу);
8. установка рабочего указателя в начало списка;
9. установка рабочего указателя в конец списка;
10. проверка пустоты списка;
11. удаление списка.

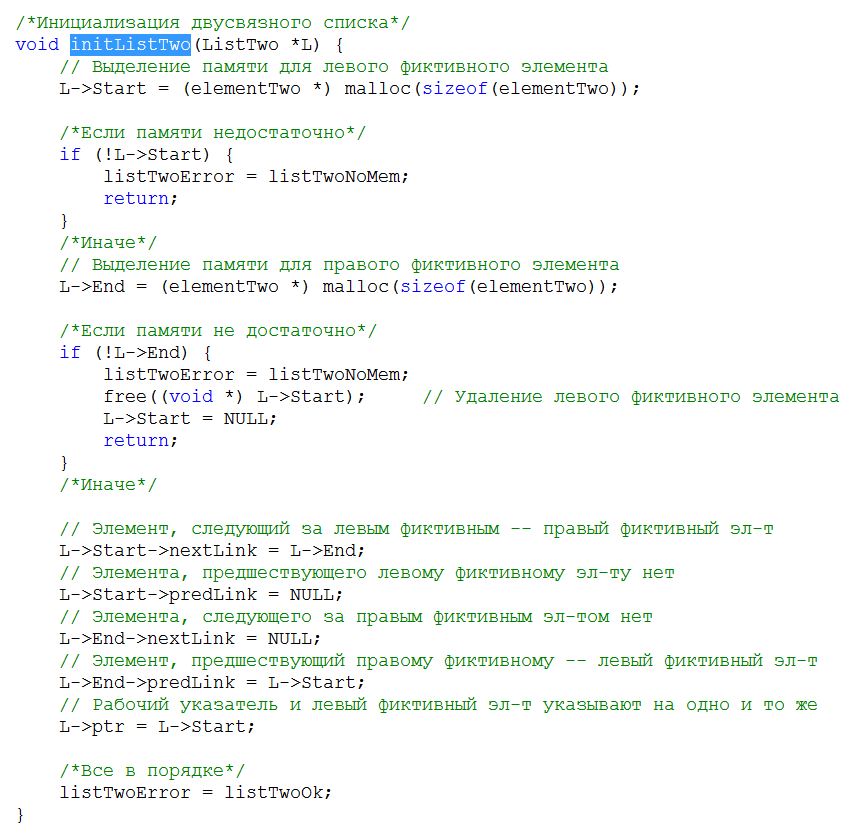
## 4.2.1 Реализация двусвязного списка

Как и с односвязным списком, в статической памяти будет располагаться дескриптор. Этот дескриптор будет в точности таким же, как и у ОЛС. Единственное отличие — два указателя.



## 4.2.2 Инициализация двусвязного списка

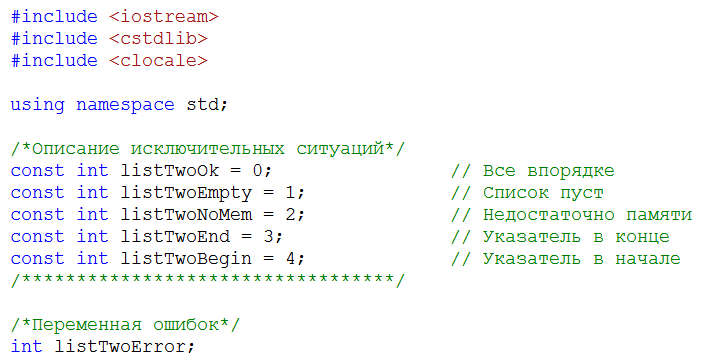
Суть инициализации состоит в том, чтобы создать фиктивные элементы, то есть выделить для них память. Интерес вызывает то, что эти указатели должны указывать друг на друга.

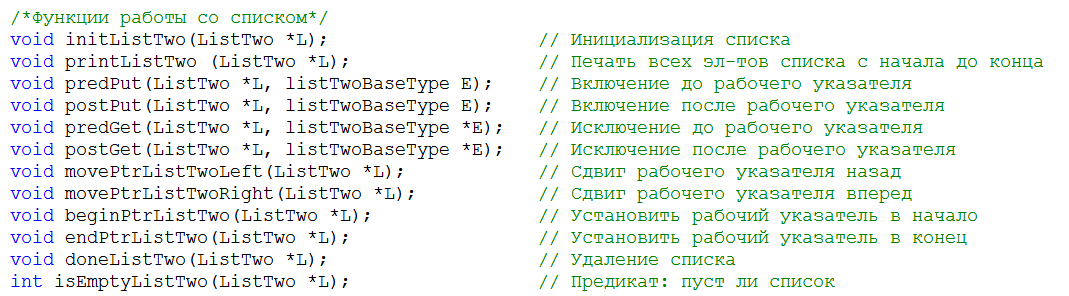


Как видите, суть такая же, как это было с односвязным списком. Остальные операции реализуются по такому же принципу, как и соответствующие операции ОЛС. Единственное, о чем нужно заботиться - это о втором указателе.

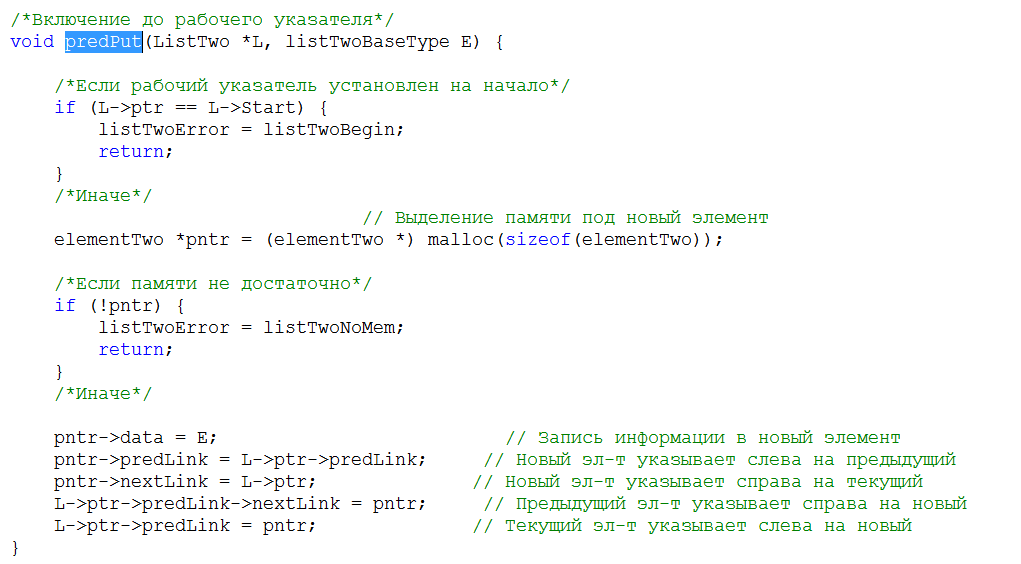
## 4.2.3 Функции для работы с двусвязным списком

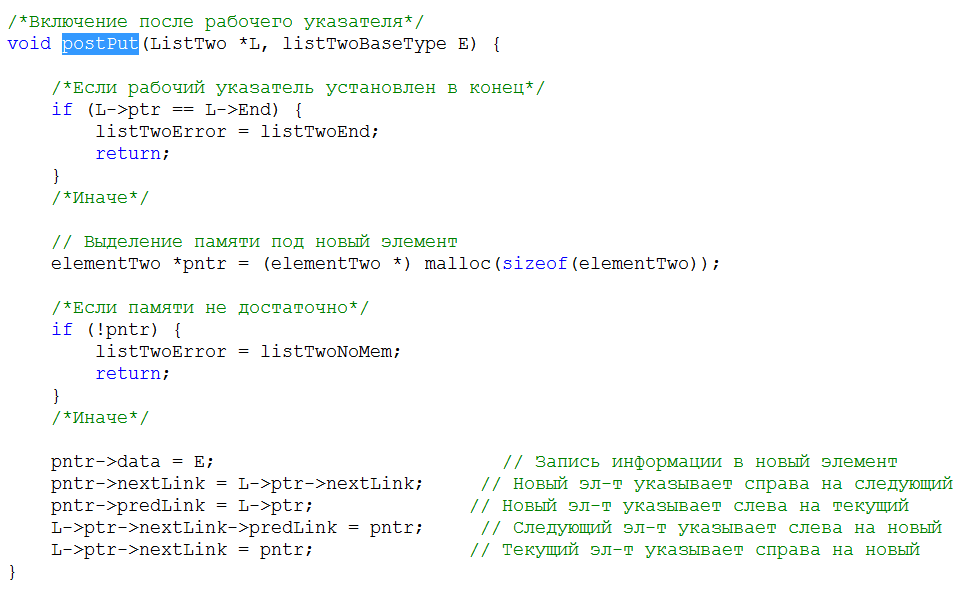
Для начала подключим библиотеки, объявим служебные (дополнительные) переменные и прототипы функций.



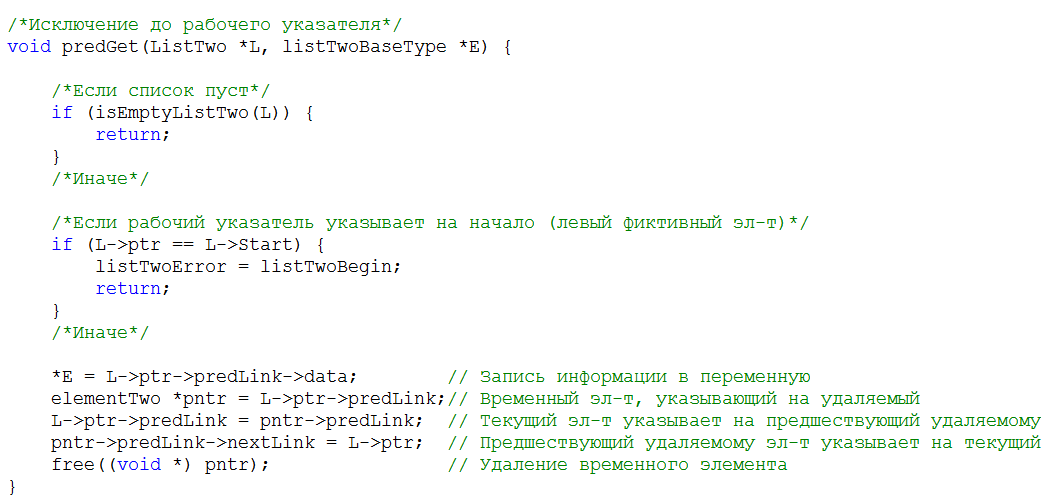


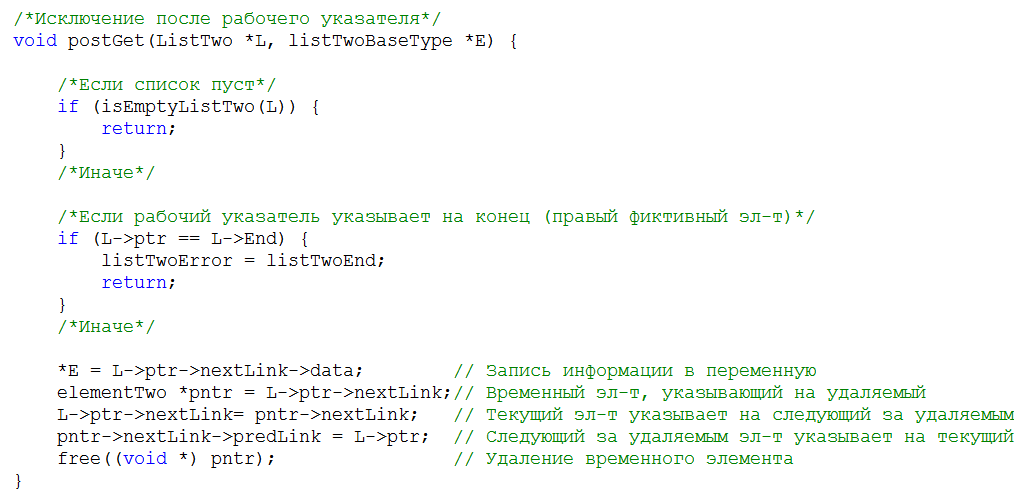
Функции для вставки элементов:



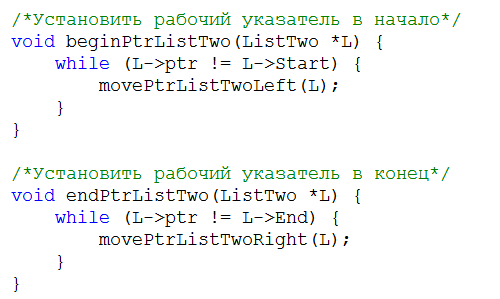


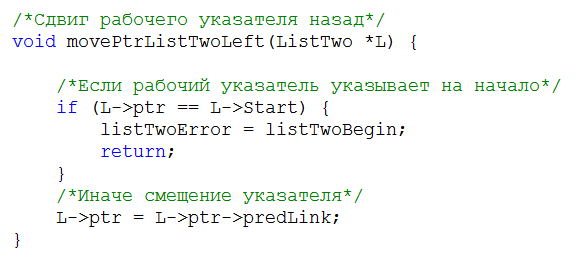
Функции для удаления элементов:

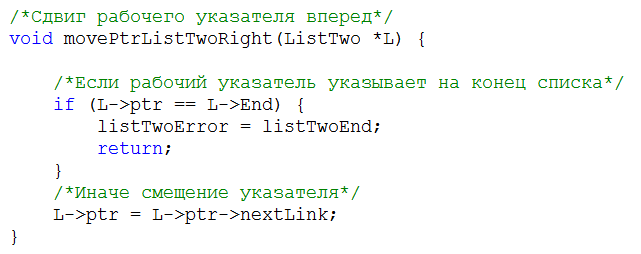




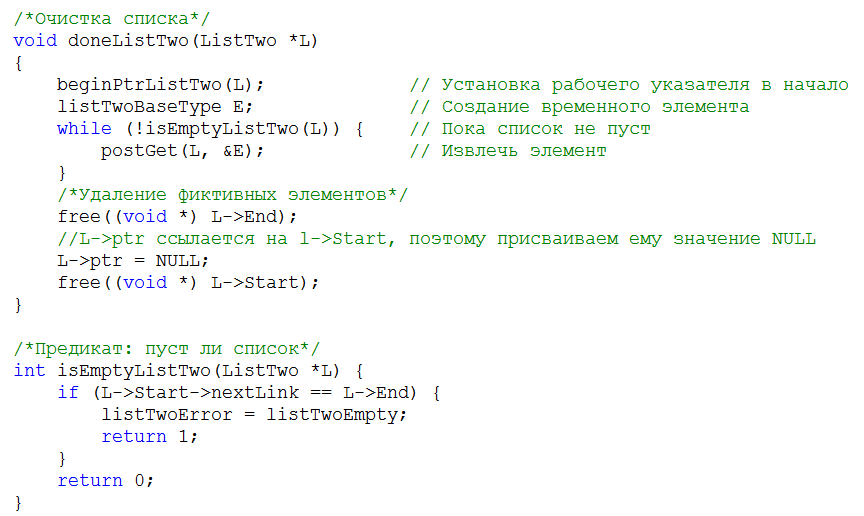
Функции сдвига рабочего указателя (***ptr***) вправо и влево относительно текущей позиции, а также в начало и в конец списка.

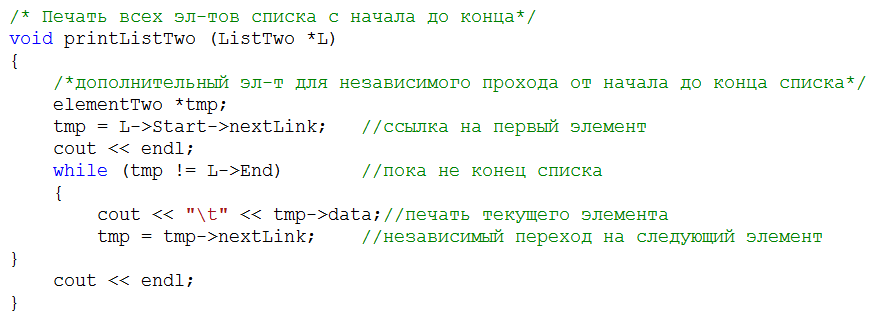






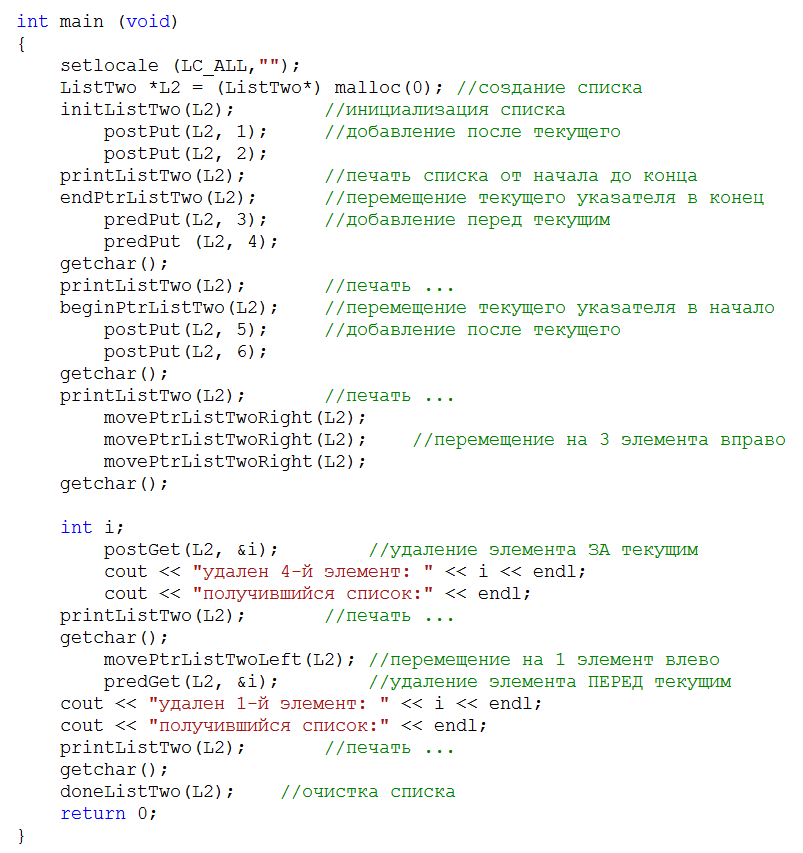
Дополнительные функции:





## 4.2.4 Тестирование функций для работы с двусвязным списком

Протестируем правильность работы описанных выше функций, написав функцию ***main()*** следующего содержания:



На рисунке 11 изображен результат тестирования программы.

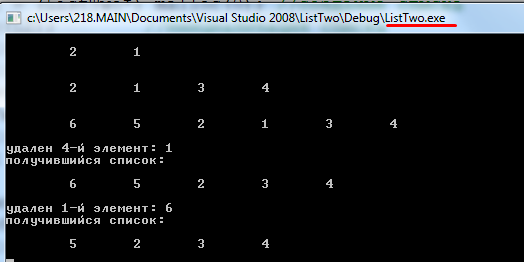


Рисунок 11 – Результат тестирования функций   
работы с двусвязным списком

ЗАДАНИЯ ДЛЯ ЛАБОРАТОРНОЙ РАБОТЫ № 2\_7

**Создайте динамический список (НЕ на основе массива) указанного типа для хранения данных, указанных в варианте задания. Напишите функции для работы со списком:**

**Задание 1.** Функции для создания нового списка и его удаления.

**Задание 2.** Функции добавления элемента в начало и после указанного элемента.

**Задание 3.** Функция удаления элемента.

**Задание 4.** Функция печати всех элементов списка.

**\*Задание 5.** Функция перестановки двух элементов списка (например, если на вход функции подается k-тый и g-тый элементы, то после срабатывания функции в списке на позиции k-того элемента окажется g-тый, а на позиции g-того – k-тый).

# ВАРИАНТЫ ЗАДАНИЙ

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **В-т** | **Задание** | **В-т** | **Задание** | **В-т** | **Задание** |
| **1.1** | линейный список  char[20] | **2.1** | линейный список  int | **3.1** | двусвязный список  double |
| **1.2** | кольцевой список  char[20] | **2.2** | кольцевой список  int | **3.2** | кольцевой список  double |
| **1.3** | двусвязный список  int | **2.3** | двусвязный список  int | **3.3** | линейный список  int |
| **1.4** | линейный список  float | **2.4** | линейный список  char[20] | **3.4** | двусвязный список  int |
| **1.5** | кольцевой список  double | **2.5** | кольцевой список  char[20] | **3.5** | кольцевой список  int |
| **1.6** | двусвязный список  int | **2.6** | двусвязный список  char[20] | **3.6** | линейный список  char[15] |
| **1.7** | линейный список  double | **2.7** | линейный список  float | **3.7** | кольцевой список  char[15] |
| **1.8** | кольцевой список  int | **2.8** | кольцевой список  float | **3.8** | двусвязный список  char[15] |
| **1.9** | двусвязный список  float | **2.9** | двусвязный список  float | **3.9** | линейный список  float |
| **1.10** | линейный список  int | **2.10** | линейный список  double | **3.10** | двусвязный список  float |
| **1.11** | кольцевой список  float | **2.11** | двусвязный список  int | **3.11** | кольцевой список  float |
| **1.12** | двусвязный список  char[15] | **2.12** | кольцевой список  float | **3.12** | линейный список  float |
| **1.13** | кольцевой список  int | **2.13** | двусвязный список  char[15] | **3.13** | двусвязный список  int |