Министерство науки и высшего образования Российской Федерации
Калужский филиал
федерального государственного бюджетного образовательного
учреждения высшего образования
**«Московский государственный технический университет
имени Н.Э. Баумана
(национальный исследовательский университет)»**
(КФ МГТУ им. Н.Э. Баумана)

Ю.С. Белов, С.С. Гришунов

РАБОТА СО СПИСОЧНЫМИ СТРУКТУРАМИ

Методические указания к выполнению лабораторной работы
по курсу «Типы и структуры данных»

Калуга – 2019

УДК 004.62
ББК 32.972.5
Б435

Методические указания составлены в соответствии с учебным планом КФ МГТУ им. Н.Э. Баумана по направлению подготовки 09.03.04 «Программная инженерия» кафедры «Программного обеспечения ЭВМ, информационных технологий».

Методические указания рассмотрены и одобрены:

- Кафедрой «Программного обеспечения ЭВМ, информационных технологий» (ИУ4-КФ) протокол № 51.4/5 от «23» января 2019 г.

Зав. кафедрой ИУ4-КФ _____ к.т.н., доцент Ю.Е. Гагарин

- Методической комиссией факультета ИУ-КФ протокол № 7 от «28» 01 2019 г.

Председатель методической комиссии факультета ИУ-КФ _____ к.т.н., доцент М.Ю. Адкин

- Методической комиссией КФ МГТУ им.Н.Э. Баумана протокол № 4 от «5» 02 2019 г.

Председатель методической комиссии КФ МГТУ им.Н.Э. Баумана _____ д.э.н., профессор О.Л. Перерва

Рецензент:
к.т.н., доцент кафедры ИУ6-КФ _____ А.Б. Лачихина

Авторы
к.ф.-м.н., доцент кафедры ИУ4-КФ _____ Ю.С. Белов
ассистент кафедры ИУ4-КФ _____ С.С. Гришунов

Аннотация

Методические указания к выполнению лабораторной работы по курсу «Типы и структуры данных» содержат сведения о динамических списковых структурах данных с целью ознакомления с работой динамических структур данных, используемых для создания программных и системных комплексов обработки информации. Рассматриваются приемы создания и обработки стека, очереди, дека, кольцевого и двунаправленного списков.

Предназначены для студентов 2-го курса бакалавриата КФ МГТУ им. Н.Э. Баумана, обучающихся по направлению подготовки 09.03.04 «Программная инженерия».

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Настоящие методические указания составлены в соответствии с программой проведения лабораторных работ по курсу «Типы и структуры данных» на кафедре «Программное обеспечение ЭВМ, информационные технологии» факультета «Информатика и управление» Калужского филиала МГТУ им. Н.Э. Баумана.

Методические указания к выполнению лабораторной работы по курсу «Типы и структуры данных» содержат сведения о видах списочных структур, их создании и обработке. Рассматриваются алгоритмы основных операций при работе со списками. Для закрепления теоретического и практического материала лабораторной работы в качестве задания приводится индивидуальный вариант для каждого студента, в котором необходимо реализовать все изученные алгоритмы работы со списками.

Предназначены для студентов 2-го курса бакалавриата КФ МГТУ им Н.Э. Баумана, обучающихся по направлению подготовки 09.03.04 «Программная инженерия».

ЦЕЛЬ И ЗАДАЧИ РАБОТЫ, ТРЕБОВАНИЯ К РЕЗУЛЬТАТАМ ЕЕ ВЫПОЛНЕНИЯ

Целью лабораторной работы является формирование практических навыков создания алгоритмов обработки списочных структур данных.

Основными задачами выполнения лабораторной работы являются:

1. Изучить основные виды списочных структур.
2. Изучить организацию списочных структур.
3. Познакомиться с основными операциями для обработки списков.
4. Изучить типовые алгоритмы решения задач с использованием списков.
5. Реализовать основные алгоритмы обработки списочных структур данных (создание, удаление, поиск, добавление и удаление элемента), а также алгоритм согласно полученному варианту.

Результатами работы являются:

- консольное приложение, написанное с применением ООП и запускаемое из командной строки;
- построенная списковая структура;
- отчет

КРАТКАЯ ХАРАКТЕРИСТИКА ОБЪЕКТА ИЗУЧЕНИЯ, ИССЛЕДОВАНИЯ

Всю совокупность данных можно разделить на две большие группы: *данные статической структуры* и *данные динамической структуры*.

Данные статической структуры характеризуются тем, что взаимное расположение и взаимосвязь элементов структуры всегда остаются постоянными.

Данные динамической структуры — это данные, внутреннее строение которых формируется по какому-либо закону, но количество элементов, их взаиморасположение и взаимосвязи могут динамически изменяться во время выполнения программы согласно закону формирования.

Динамическая структура данных характеризуется тем что:

- она не имеет имени;
- ей выделяется память в процессе выполнения программы;
- количество элементов структуры может не фиксироваться;
- размерность структуры может меняться в процессе выполнения программы;
- в процессе выполнения программы может меняться характер взаимосвязи между элементами структуры.

Каждой динамической структуре данных сопоставляется *статическая переменная* типа указатель (ее значение – адрес этого объекта), посредством которой осуществляется доступ к динамической структуре.

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*.

Во многих задачах требуется использовать данные, у которых конфигурация, размеры и состав могут меняться в процессе выполнения программы. Для их представления используют динамические информационные структуры. К таким структурам относят:

- однонаправленные (односвязные) списки;
- двунаправленные (двусвязные) списки;
- циклические списки;
- стек;
- дек;
- очередь;
- бинарные деревья.

Они отличаются способом связи отдельных элементов и/или допустимыми операциями. Динамическая структура может занимать несмежные участки оперативной памяти.

Элемент динамической структуры состоит из двух полей:

информационного поля (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.;

адресного поля (поля связок), в котором содержатся один или несколько указателей, связывающий данный элемент с другими элементами структуры;

Объявление элемента динамической структуры данных выглядит следующим образом:

```
struct имя_типа
{
      информационное поле;
      адресное поле;
};
```

Например (рис. 1):

```
struct TNode
{
      int Data; //информационное поле
      TNode *Next; //адресное поле
};
```

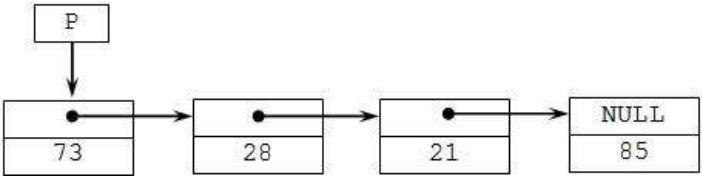

Рис. 1. Схематичное представление динамической структуры

Информационных и адресных полей может быть, как одно, так и несколько.

Однонаправленный (односвязный) список

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*. Из всего многообразия связанных списков можно выделить следующие основные:

- однонаправленные (односвязные) списки;
- двунаправленные (двусвязные) списки;
- циклические (кольцевые) списки.

Однонаправленный (односвязный) список – это структура данных, представляющая собой последовательность элементов, в каждом из которых хранится значение и указатель на следующий элемент списка.

В последнем элементе указатель на следующий элемент равен NULL (рис. 2).

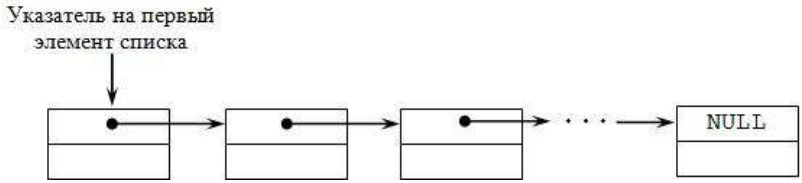

Рис. 2. Линейный однонаправленный список

Основными операциями, осуществляемыми с однонаправленными списками, являются:

- создание списка;
- печать (просмотр) списка;
- вставка элемента в список;
- удаление элемента из списка;
- поиск элемента в списке
- проверка пустоты списка;
- удаление списка.

Создание однонаправленного списка

Для того чтобы создать список, нужно создать сначала первый элемент списка, а затем при помощи функции добавить к нему остальные элементы. При относительно небольших размерах списка наиболее изящно и красиво использование рекурсивной функции. Добавление может выполняться как в начало, так и в конец списка.

Печать однонаправленного списка

Операция печати списка заключается в последовательном просмотре всех элементов списка и выводе их значений на экран. Для обработки списка организуется функция, в которой нужно переставлять указатель на следующий элемент списка до тех пор, пока указатель не станет равен NULL, то есть будет достигнут конец списка.

Вставка элемента в однонаправленный список

В динамические структуры легко добавлять элементы, так как для этого достаточно изменить значения адресных полей. Вставка первого и последующих элементов списка отличаются друг от друга. Поэтому в функции, реализующей данную операцию, сначала осуществляется проверка, на какое место вставляется элемент. Далее реализуется соответствующий алгоритм добавления (рис. 3).

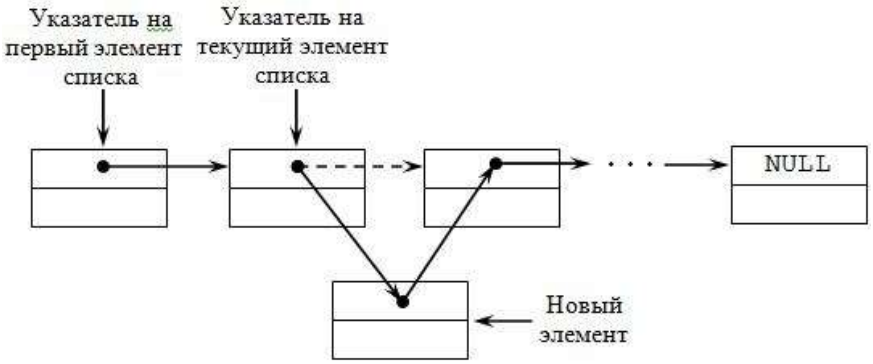

Рис. 3. Вставка элемента в однонаправленный список

Удаление элемента из однонаправленного списка

Из динамических структур можно удалять элементы, так как для этого достаточно изменить значения адресных полей. Операция удаления элемента однонаправленного списка осуществляет удаление элемента, на который установлен указатель текущего элемента. После удаления указатель текущего элемента устанавливается на предшествующий элемент списка или на новое начало списка, если удаляется первый.

Алгоритмы удаления первого и последующих элементов списка отличаются друг от друга. Поэтому в функции, реализующей данную операцию, осуществляется проверка, какой элемент удаляется. Далее реализуется соответствующий алгоритм удаления (рис. 4).

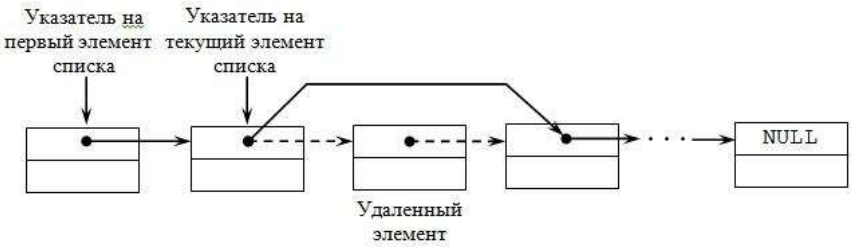

Рис. 4. Удаление элемента из однонаправленного списка

Поиск элемента в однонаправленном списке

Операция поиска элемента в списке заключается в последовательном просмотре всех элементов списка до тех пор, пока текущий элемент не будет содержать заданное значение или пока не будет достигнут конец списка. В последнем случае фиксируется отсутствие искомого элемента в списке (функция принимает значение false).

Удаление однонаправленного списка

Операция удаления списка заключается в освобождении динамической памяти. Для данной операции организуется функция, в которой нужно переставлять указатель на следующий элемент списка до тех пор, пока указатель не станет равен NULL, то есть не будет достигнут конец списка.

Двунаправленный (двусвязный) список

Двунаправленный список – это структура данных, состоящая из последовательности элементов, каждый из которых содержит информационную часть и два указателя на соседние элементы (рис. 5). При этом два соседних элемента должны содержать взаимные ссылки друг на друга.

В таком списке каждый элемент (кроме первого и последнего) связан с предыдущим и следующим за ним элементами. Каждый элемент *двунаправленного* списка имеет два поля с указателями: одно поле содержит ссылку на следующий элемент, другое поле – ссылку на предыдущий элемент и третье поле – информационное. Наличие

ссылок на следующее звено и на предыдущее позволяет двигаться по списку от каждого звена в любом направлении: от звена к концу списка или от звена к началу списка, поэтому такой список называют двунаправленным.

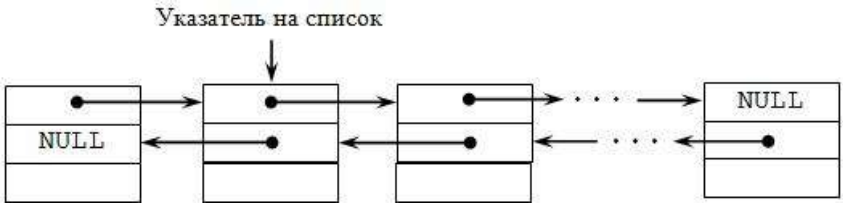

Рис. 5. Двунаправленный список

Описание простейшего элемента такого списка выглядит следующим образом:

```
struct имя_типа
{
      информационное поле;
      адресное поле 1;
      адресное поле 2;
};
```

где *информационное поле* – это поле любого, ранее объявленного или стандартного, типа;

адресное поле 1 – это указатель на объект того же типа, что и определяемая структура, в него записывается адрес *следующего элемента списка*;

адресное поле 2 – это указатель на объект того же типа, что и определяемая структура, в него записывается адрес *предыдущего элемента списка*.

Операции для работы с двусвязным списком

Следует отметить, что перечень операций для работы с двусвязным списком аналогичен перечню операций для односвязного списка.

Отдельно рассмотрим операции вставки, удаления и поиска элемента в списке.

Вставка элемента в двунаправленный список

В динамические структуры легко добавлять элементы, так как для этого достаточно изменить значения адресных полей. Операция вставки реализовывается аналогично функции вставки для однонаправленного списка, только с учетом особенностей *двунаправленного* списка (рис. 6).

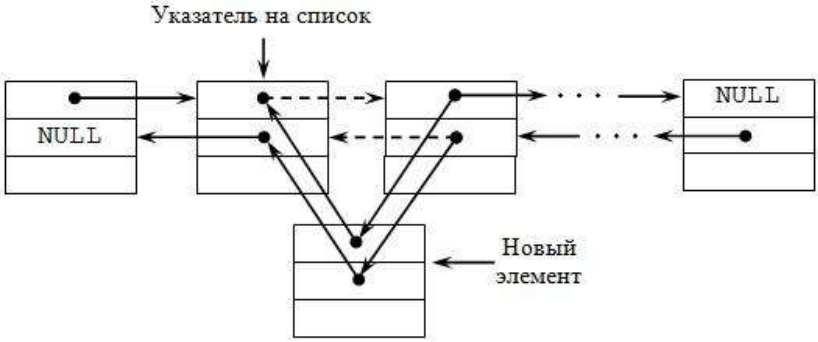

Рис. 6. Добавление элемента в двунаправленный список

Удаление элемента из двунаправленного списка

Из динамических структур можно удалять элементы, так как для этого достаточно изменить значения адресных полей. Операция удаления элемента из двунаправленного списка осуществляется во многом аналогично удалению из однонаправленного списка (рис. 7).

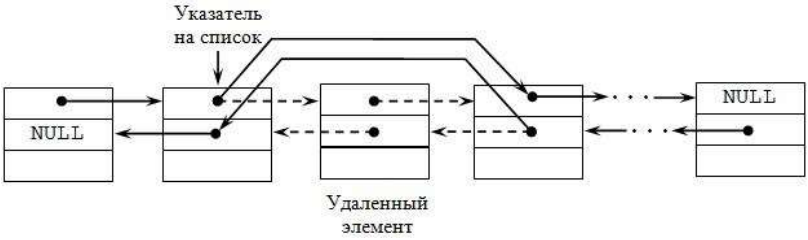

Рис. 7. Удаление элемента из двунаправленного списка

Поиск элемента в двунаправленном списке

Операция поиска элемента в двунаправленном списке реализуется абсолютно аналогично соответствующей функции для однонаправленного списка. Поиск элемента в двунаправленном списке можно вести:

а) просматривая элементы от начала к концу списка;

б) просматривая элементы от конца списка к началу;

в) просматривая список в обоих направлениях одновременно: от начала к середине списка и от конца к середине (учитывая, что элементов в списке может быть четное или нечетное количество).

Очередь, стек и дек

Стек

В списках доступ к элементам происходит посредством адресации, при этом доступ к отдельным элементам не ограничен. Но существуют также и такие списковые структуры данных, в которых имеются ограничения доступа к элементам. Одним из представителей таких списковых структур является стековый список или просто стек.

Стек (англ. stack – стопка) – это структура данных, в которой новый элемент всегда записывается в ее начало (вершину) и очередной читаемый элемент также всегда выбирается из ее начала (рис. 8). В стеках используется метод доступа к элементам LIFO (Last Input – First Output, "последним пришел – первым вышел"). Чаще всего принцип работы стека сравнивают со стопкой тарелок: чтобы взять вторую сверху, нужно сначала взять верхнюю.

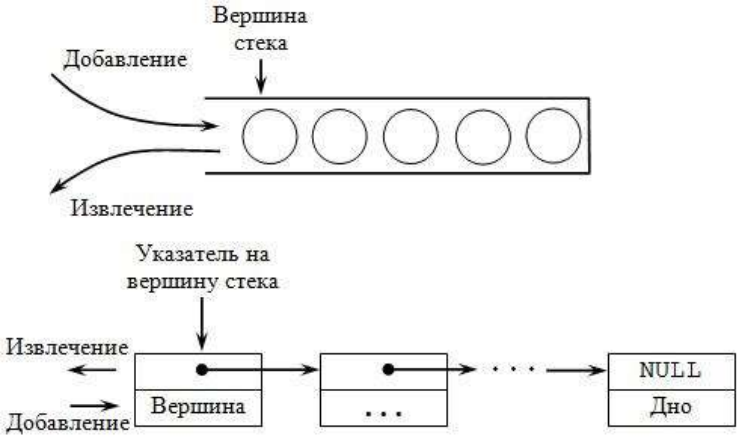

Рис. 8. Стек и его организация

Стек – это список, у которого доступен один элемент (одна позиция). Этот элемент называется вершиной стека. Взять элемент можно только из вершины стека, добавить элемент можно только в вершину стека. Например, если записаны в стек числа 1, 2, 3, то при последующем извлечении получим 3,2,1.

Описание стека выглядит следующим образом:

```
struct имя_типа
{
      информационное поле;
      адресное поле;
};
```

где *информационное поле* – это поле любого ранее объявленного или стандартного типа;

адресное поле – это указатель на объект того же типа, что и определяемая структура, в него записывается адрес следующего элемента стека.

Например:

```
struct list
{
      type  pole1;
      list *pole2;
} stack;
```

Стек как динамическую структуру данных легко организовать на основе линейного списка. Поскольку работа всегда идет с заголовком стека, то есть не требуется осуществлять просмотр элементов, удаление и вставку элементов в середину или конец списка, то достаточно использовать экономичный по памяти линейный однонаправленный список. Для такого списка достаточно хранить указатель *вершины стека*, который указывает на первый элемент списка. Если стек пуст, то списка не существует, и указатель принимает значение NULL.

Описание элементов стека аналогично описанию элементов линейного однонаправленного списка. Поэтому объявим стек через объявление линейного однонаправленного списка:

```
struct Stack
{
      Single_List *Top; //вершина стека
};
. . . . . .
Stack *Top_Stack;    //указатель на вершину стека
```

Основные операции, производимые со стеком:

- создание стека;
- печать (просмотр) стека;
- добавление элемента в вершину стека;
- извлечение элемента из вершины стека;
- проверка пустоты стека;
- очистка стека.

Очередь

Очередь – это структура данных, представляющая собой последовательность элементов, образованная в порядке их поступления. Каждый новый элемент размещается в конце очереди; элемент, стоящий в начале очереди, выбирается из нее первым. В очереди используется принцип доступа к элементам *FIFO* (*First Input – First Output*, "пер-вый пришёл – первый вышел") (рис. 9). В очереди доступны два элемента (две позиции): *начало очереди* и *конец очереди*. Поместить элемент можно только в конец очереди, а взять элемент только из ее начала. Примером может служить обыкновенная очередь в магазине.

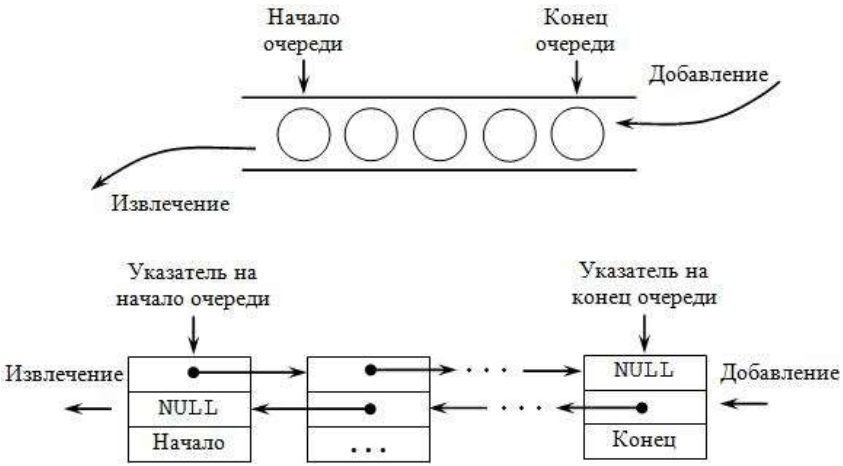

Рис. 9. Очередь и ее организация

Описание очереди выглядит следующим образом:

```
struct имя_типа
{
      информационное поле;
      адресное поле1;
      адресное поле2;
};
```

где *информационное поле* – это поле любого, ранее объявленного или стандартного, типа;

адресное поле1, *адресное поле2* – это указатели на объекты того же типа, что и определяемая структура, в них записываются адреса первого и следующего элементов очереди.

Очередь как динамическую структуру данных легко организовать на основе линейного списка. Поскольку работа идет с обоими концами очереди, то предпочтительно будет использовать линейный двунаправленный список. Хотя для работы с таким списком достаточно иметь один указатель на любой элемент списка, здесь целесообразно хранить два указателя – один на начало списка (откуда извлекаем элементы) и один на конец списка (куда добавляем элементы). Если

очередь пуста, то списка не существует, и указатели принимают значение NULL.

Основные операции, производимые с очередью:

- создание очереди;
- печать (просмотр) очереди;
- добавление элемента в конец очереди;
- извлечение элемента из начала очереди;
- проверка пустоты очереди;
- очистка очереди.

Дек

Дек является особым видом очереди.

Дек (англ. deque – аббревиатура от double-ended queue, двухсторонняя очередь) – это структура данных, представляющая собой последовательность элементов, в которой можно добавлять и удалять в произвольном порядке элементы с двух сторон (рис. 10). Первый и последний элементы дека соответствуют входу и выходу дека.

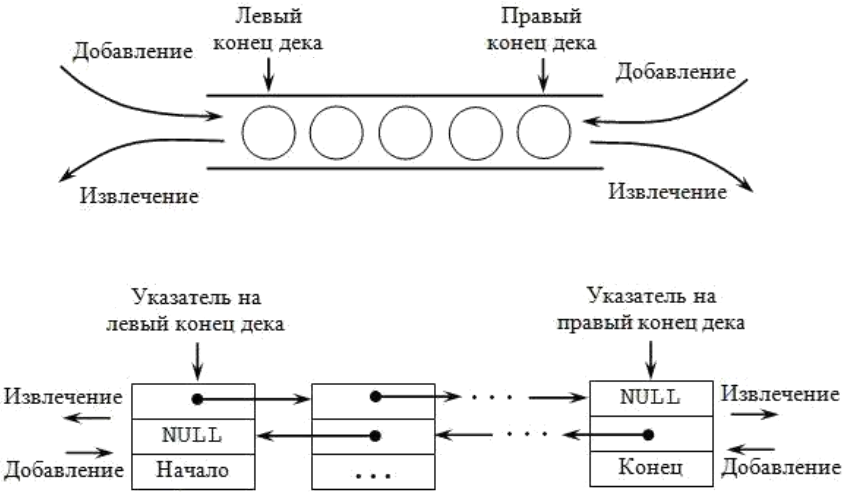

Рис. 10. Дек и его организация

Частные случаи дека – это ограниченные деки:

- дек с ограниченным входом – из конца дека можно только извлекать элементы;
- дек с ограниченным выходом – в конец дека можно только добавлять элементы.

Данная структура является наиболее универсальной из рассмотренных выше линейных структур. Накладывая дополнительные ограничения на операции с началом и/или концом *дека*, можно осуществлять моделирование стека и очереди.

Однако применительно к деку целесообразно говорить не о начале и конце как в очереди, а о левом и правом конце.

Описание элементов *дека* аналогично описанию элементов линейного двунаправленного списка. Поэтому объявим *дек* через объявление линейного *двунаправленного* списка:

- создание дека;
- печать (просмотр) дека;
- добавление элемента в левый конец дека;
- добавление элемента в правый конец дека;
- извлечение элемента из левого конца дека;
- извлечение элемента из правого конца дека;
- проверка пустоты дека;
- очистка дека

Циклический (кольцевой) список

Циклический (кольцевой) список – это структура данных, представляющая собой последовательность элементов, последний элемент которой содержит указатель на первый элемент списка, а первый (в случае двунаправленного списка) – на последний.

Основная особенность такой организации состоит в том, что в этом списке нет элементов, содержащих пустые указатели, и, следовательно, нельзя выделить крайние элементы.

Циклические списки, так же как и линейные, бывают однонаправленными и *двунаправленными*.

Циклический однонаправленный список похож на линейный однонаправленный список, но его последний элемент содержит указатель, связывающий его с первым элементом (рис. 11).

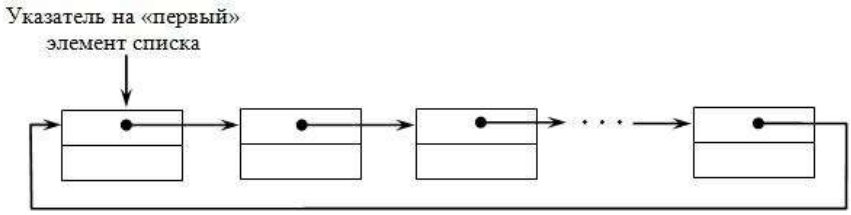

Рис. 11. Циклический однонаправленный список

Для полного обхода такого списка достаточно иметь указатель на произвольный элемент, а не на первый, как в линейном однонаправленном списке. Понятие "первого" элемента здесь достаточно условно и не всегда требуется. Хотя иногда бывает полезно выделить некоторый элемент как "первый" путем установки на него специального указателя. Это требуется, например, для предотвращения "зацикливания" при просмотре списка.

Основные операции, осуществляемые с циклическим однонаправленным списком:

- создание списка;
- печать (просмотр) списка;
- вставка элемента в список;
- удаление элемента из списка;
- поиск элемента в списке;
- проверка пустоты списка; удаление списка.

ТЕСТИРОВАНИЕ И РЕАЛИЗАЦИЯ СПИСКОВ

Рассмотрим реализацию однонаправленного списка на примере списка, представленного как отдельная сущность в виде класса.

```
struct Node
{
    int data;
    Node * next;
};

class SingleList
{
    Node *top;
    int size;
public:
    SingleList() { top = NULL; size = 0; }
    Node * GetTop() { return top; }
    int GetSize() { return size; };
    void AddItem(int); void
    DeleteItem(int);
    bool IsItem(int);
    bool IsEmpty();
    void PrintList();
    void PrintListRecurrently(Node *);
    void DeleteList(Node *);
};
```

Далее рассмотрим подробнее рассмотрим методы класса.

```
void SingleList :: AddItem(int _data)
{
    Node * temp;
    temp = new Node;
    temp->data = _data;
    temp->next = top;
    top = temp;
    size++;
}
void SingleList :: PrintList()
{
   cout << "\nThe size is "<< size << endl;
```

```
    Node * temp = top;
    if (!size)
    {
          throw SLException(SLException::EMPTY_LIST,
"empty list!");
    }
    while (temp != NULL)
    {
          cout << "-> " << temp->data << "\t: " <<
temp->next << endl;
          temp = temp->next;
    }
    cout << endl << endl;
}
void SingleList :: PrintListRecurrently(Node *_top) {
    if (!size)
    {
          throw SLException(SLException::EMPTY_LIST,
"empty list!");
    }
    if (_top != NULL)
    {
          cout << _top->data << "\t";
          PrintListRecurrently(_top->next);
    }
}

void SingleList :: DeleteItem(int index)
{
    if ((index < 0) || (index > size))
    {
          throw SLExcep-
tion(SLException::OUT_OF_INDEX, "out of index!");
    }
    if (!size)
    {
          throw SLException(SLException::EMPTY_LIST,
"empty list!");
}

    Node * temp, * prev, * ptrNode;
    if (index == 1)
    {
```

```
            temp = top;
            top = top->next;
            delete temp;
            size--;
      }
      else if (index == size)
            {
                  prev = top;
                  while (prev->next->next != NULL)
                  {
                        prev = prev->next;
                  }
                  temp = prev->next;
                  prev->next = NULL;
                  delete temp;
                  size--;
            }
            else
            {
                  prev = top;
                  int i = 1;
                  while (i < (index-1))
                  {
                        prev = prev->next;
                        i++;
                  }
                  temp = prev->next;
                  prev->next = prev->next->next;
                  delete temp;
                  size--;
            }
}
bool SingleList :: IsItem(int source)
{
      Node * temp = top;
      while (temp != NULL)
      {
            if (temp->data == source)
            {
                  break;
            }
              temp = temp->next;
        }
        return (temp != NULL) ? 1 : 0;
```

```
  }
  bool SingleList :: IsEmpty()
  {
      return (top == NULL) ? 1 : 0;
  }
  void SingleList :: DeleteList(Node *_top)
  {
      cout << "inside deleting:\t";
      if (_top->next != NULL) {
            cout << "-> " << _top->data << "\t: " <<
              top->next << endl;
            DeleteList(_top->next);
      }
      cout << "\t\t" << _top->data << "\t: " <<
          _top->next << endl;
      _top->next = NULL;
      size--;
      delete _top;
}
```

А вот как выглядит создание объекта `SingleList` в основной программе:

```
SingleList *list = new SingleList();
```

После создания объекта в основной программе можно продолжать с ним работать посредством вызовов его методов. Например, заполнить список можно следующим образом:

```
  int size, item;
  cout << "Enter the number of items:\t";
  cin >> _size;
  for (int i = 0; i < size; i++)
  {
      cout << "Enter an item:\t";
      cin >> item;
      list ->AddItem(item);
  }
```

ЗАДАНИЕ НА ЛАБОРАТОРНУЮ РАБОТУ

1. Разработать консольное приложение, написанное с помощью объектно-ориентированной технологии. Индивидуальное задание предусмотрено вариантом, который назначает преподаватель.
2. Приложение необходимо запускать для демонстрации из командной строки с указанием названий приложения и трех файлов:
   - все входные данные (например, последовательности чисел, коэффициенты многочленов и т.д.) считать из первого файла;
   - все выходные данные записать во второй файл;
   - все возникшие ошибки записать в третий файл – файл ошибок.
3. Все основные сущности приложения представить в виде отдельных классов.
4. Необходимо предусмотреть пользовательское меню, содержащее набор команд всех основных операций для работы со списком, а также команду для запуска индивидуального задания.
5. В приложении также должны быть учтены все критические ситуации, обработанные с помощью класса исключений.

ВАРИАНТЫ ЗАДАНИЙ

1. Даны натуральное число n, действительные числа a_1, a_2, ..., a_n. Если последовательность a_1, a_2, ..., a_n упорядочена по возрастанию, то оставить ее без изменения. Иначе получить последовательность a_n, ..., a_1 (для решения задачи использовать односвязный список).

2. Даны натуральное число n, действительные числа x_1, x_2, ..., x_n. Вычислить $x_1*x_n + x_2*x_{n-1} + ... + x_nx_1$ (для решения задачи использовать двусвязный список).

3. Даны натуральное число n, действительные числа x_1, x_2, ..., x_n. Вычислить $(x_1+x_n)*(x_2+x_{n-1})*...*(x_n+x_1)$ (для решения задачи использовать двусвязный список).

4. Даны натуральное число n, действительные числа x_1, x_2, ..., x_n. Вычислить $(x_1+x_2+2x_n)*(x_2+x_3+2x_{n-1})*...*(x_{n-1}+x_n+2x_2)$ (для решения задачи использовать двусвязный список).

5. Даны натуральное число n, действительные числа a_1, a_2, ..., a_n. Преобразовать последовательность a_1, a_2, ..., a_n, расположив вначале отрицательные члены, а затем - неотрицательные. При этом порядок как отрицательных, так и неотрицательных чисел сохраняется прежним (для решения задачи использовать линейный динамический список).

6. Даны натуральное число n, действительные числа a_1, a_2, ..., a_n. Преобразовать последовательность a_1, a_2, ..., a_n, расположив вначале отрицательные члены, а затем - неотрицательные. При этом порядок отрицательных чисел изменяется на обратный, а порядок неотрицательных чисел сохраняется прежним (для решения задачи использовать линейный динамический список).

7. Даны натуральное число n, действительные числа a_1, a_2, ..., a_n. Преобразовать последовательность a_1, a_2, ..., a_n, расположив вначале

отрицательные члены, а затем - неотрицательные. При этом порядок отрицательных чисел сохраняется прежним, а порядок неотрицательных чисел изменяется на обратный (для решения задачи использовать линейный динамический список).

8. Даны натуральное число n, действительные числа a_1, a_2, ..., a_n. Преобразовать последовательность a_1, a_2, ..., a_n, расположив вначале отрицательные члены, а затем - неотрицательные. При этом порядок как отрицательных, так и неотрицательных чисел изменяется на обратный (для решения задачи использовать линейный динамический список).

9. Даны натуральное число n, действительные числа a_1, a_2, ..., a_{2n}. Получить: $(a_1-a_{2n})*(a_3-a_{2n-2})*(a_5-a_{2n-4})*...*(a_{2n-1}-a_2)$ (для решения задачи использовать линейный динамический список).

10. Даны натуральное число n, действительные числа a_1, a_2, ..., a_{2n}. Получить: $a_1*a_{2n} + a_2*a_{2n-1} +...+ a_n*a_{n+1}$) (для решения задачи использовать линейный динамический список).

11. Даны натуральное число n, действительные числа a_1, a_2, ..., a_{2n}. Получить: $\min(a_1+a_{n+1}, a_2+a_{n+2}, ..., a_n+a_{2n})$ (для решения задачи использовать линейный динамический список).

12. Даны натуральное число n, действительные числа a_1, a_2, ..., a_{2n}. Получить: $\max(\min(a_1,a_{2n}),\min(a_2,a_{2n-1}),...,\min(a_n,a_{n+1}))$ (для решения задачи использовать линейный динамический список).

13. Даны натуральное число n, символы s_1, s_2, ..., s_n. Получить последовательность символов, содержащую только последние вхождения каждого символа с сохранением взаимного порядка этих вхождений (для решения задачи использовать линейный динамический список).

14. Даны натуральное число n, символы s_1, s_2, ..., s_n. Получить последовательность символов, содержащую только первые вхождения каждого символа с сохранением взаимного порядка этих вхождений (для решения задачи использовать линейный динамический список).

15. Даны натуральное число n, действительные числа x1, x2, ..., xn. Вычислить (x1-xn)*(x2-xn-1)*...*(xn-x1) (для решения задачи использовать двусвязный список).

16. Задан текст, состоящий из строк, разделенных пробелом и оканчивающийся точкой. подсчитать количество вхождений четных десятичных цифр в каждую строку текста (для решения задачи использовать линейный односвязный динамический список).

17. Задан текст, состоящий из строк, разделенных пробелом и оканчивающийся точкой. подсчитать количество вхождений нечетных десятичных цифр в каждую строку текста (для решения за-дачи использовать линейный односвязный динамический спи-сок).

18. Задан текст, состоящий из строк, разделенных пробелом и оканчивающийся точкой. Удалить все вхождения заданного символа из текста (для решения задачи использовать линейный односвязный динамический список).

19. Задан текст, состоящий из строк, разделенных пробелом и оканчивающийся точкой. После последнего вхождения каждой гласной латинской буквы в строку текста вставить цифру, изображающую число вхождений этой гласной в данную строку (в строке содержится не более девяти одинаковых гласных) (для решения задачи использовать линейный односвязный динамический список).

20. Для заданных полиномов Pn(x) и Qn(x) найти сумму полиномов P и Q. Каждый полином представить в виде списка (для решения задачи использовать линейный односвязный динамический список).

21. Дана последовательность символов s1, s2, . . . , sn (n >= 2 и заранее неизвестно). Получить те символы, принадлежащие последовательности, которые входят в нее по одному разу (для решения задачи использовать линейный двусвязный динамический список).

22. Дана последовательность символов s1, s2, . . ., sn (n >= 2 и заранее неизвестно). Получить повторяющиеся символы (для решения задачи использовать линейный двусвязный динамический список).

23. Дана последовательность символов, оканчивающаяся точкой. Подсчитать количество символов, у которых левый сосед больше правого соседа (первый и последний элемент считать соседями) (для решения задачи использовать линейный двусвязный динамический список).

24. Дана последовательность символов, оканчивающаяся точкой. Удалить все символы, у которых равные соседи (первый и последний символы считать соседями) (для решения задачи использовать линейный двусвязный динамический список).

25. Дана последовательность латинских букв, оканчивающаяся точкой. Среди букв есть специальный символ, появление которого означает отмену предыдущей буквы; k знаков подряд отменяют k предыдущих букв, если такие есть. Учитывая вхождение этого символа преобразовать последовательность (для решения задачи использовать линейный двусвязный динамический список).

26. Дан многочлен P(x) произвольной степени с целыми коэффициентами, причем его одночлены могут быть не упорядочены по степеням x, а одночлены с одинаковой степенью могут повторяться (например, $-75x + 8x^4 - x^2 + 6x^4 - 5 - x$). Привести подобные члены в этом многочлене и расположить одночлены по убыванию степеней x

(для решения задачи использовать линейный двусвязный динамический список).

27. Дан многочлен P(x) произвольной степени с целыми коэффициентами, причем его одночлены могут быть не упорядочены по степеням x, а одночлены с одинаковой степенью могут повторяться (например, -75x + 8x4 - x2 + 6x4 - 5 - x). Привести подобные члены в этом многочлене и расположить одночлены по возрастанию степеней x (для решения задачи использовать линейный двусвязный динамический список).

28. Даны действительные числа x1, x2, ..., xn (n >= 2 и заранее неизвестно). Вычислить: x1*xn + x2*xn-1 + ... + xn*x1 (для решения задачи использовать линейный двусвязный динамический список).

29. Даны действительные числа x1, x2, ..., xn (n >= 2 и заранее неизвестно). Вычислить: (x1 + xn)*(x2 + xn-1) ... (xn + x1) (для решения задачи использовать линейный двусвязный динамический список).

30. Даны действительные числа x1, x2, ..., xn (n >= 2 и заранее неизвестно). Вычислить: (x1 + x2 + 2xn)*(x2 + x3 + 2xn-1)...(xn-1 + xn + 2x2) (для решения задачи использовать линейный двусвязный динамический список).

31. Даны действительные числа y1, y2, ..., yn (n >= 2 и заранее неизвестно). Получить последовательность: y1, y2, ..., yn, y1 , ..., yn (для решения задачи использовать линейный двусвязный динамический список).

32. Даны действительные числа y1, y2, ..., yn (n >= 2 и заранее неизвестно). Получить последовательность: y1, y2, ..., yn, yn, ..., y1 (для решения задачи использовать линейный двусвязный динамический список).

33. Даны действительные числа y1, y2, ..., yn (n >= 2 и заранее неизвестно). Получить последовательность: yn, yn-1, ..., y1, y1, ..., yn (для решения задачи использовать линейный двусвязный динамический список).

34. Даны действительные числа a1, a2, ..., a2n (n >= 2 и заранее неизвестно). Вычислить: a1*a2n + a2*a2n-1 + . . . + an*an+1 (для решения задачи использовать линейный двусвязный динамический список).

35. Даны действительные числа a1, a2, ..., a2n (n >= 2 и заранее неизвестно). Вычислить: min(a1 + an+1, a2 + an+2, ..., an + a2n) (для решения задачи использовать линейный двусвязный динамический список).

36. Даны действительные числа a1, a2, ..., a2n (n >= 2 и заранее неизвестно). Вычислить: max(min(a_1, a_{2n}), min(a_3, a_{2n-2}), ..., min(a_{2n-1}, a_2)) (для решения задачи использовать линейный двусвязный динамический список).

КОНТРОЛЬНЫЕ ВОПРОСЫ И ЗАДАНИЯ

1. Дайте определение понятию типа указатель.
2. Перечислите операции над указателями.
3. Раскройте понятие статической и динамической переменной.
4. Изложите концепцию создания и уничтожения динамического объекта. Перечислите операции над динамическим объектом.
5. Сформулируйте понятие списка.
6. Найдите и охарактеризуйте различия списка и массива.
7. Раскройте понятие кольцевого односвязного списка. Опишите способ задания кольцевого односвязного списка.
8. Сформулируйте определение понятия линейного *n*-связного списка. Изложите способ задания *n*-связного списка.
9. Дайте определение стека и очереди.
10. Назовите основные операции над стеком и очередью.

ФОРМА ОТЧЕТА ПО ЛАБОРАТОРНОЙ РАБОТЕ

На выполнение лабораторной работы отводится 3 занятия (6 академических часов: 5 часов на выполнение и сдачу практического задания и 1 час на подготовку отчета).

Номер варианта студента назначается индивидуально преподавателем.

Номер варианта студенту выдается преподавателем.

Отчет на защиту предоставляется в печатном виде.

Структура отчета (на отдельном листе(-ах)): титульный лист, формулировка задания, этапы выполнения работы, результаты выполнения, выводы.

ОСНОВНАЯ ЛИТЕРАТУРА

1. Алексеев В.Е. Графы и алгоритмы. Структуры данных. Модели вычислений [Электронный ресурс]/ В.Е. Алексеев, В.А. Таланов. — Электрон. текстовые данные. — М.: Интернет-Университет Информационных Технологий (ИНТУИТ), 2016. — 153 с. — Режим доступа: http://www.iprbookshop.ru/52186.html

2. Вирт Никлаус. Алгоритмы и структуры данных [Электронный ресурс]/ Никлаус Вирт— Электрон. текстовые данные. — Саратов: Профобразование, 2017. — 272 с.— Режим доступа: http://www.iprbookshop.ru/63821.html

3. Самуйлов С.В. Алгоритмы и структуры обработки данных [Электронный ресурс]: учебное пособие/ С.В. Самуйлов. — Электрон. текстовые данные. — Саратов: Вузовское образование, 2016. — 132 с.— Режим доступа: http://www.iprbookshop.ru/47275.html

ДОПОЛНИТЕЛЬНАЯ ЛИТЕРАТУРА

4. Костюкова Н.И. Графы и их применение [Электронный ресурс]/ Н.И. Костюкова. — Электрон. текстовые данные. — М.: Интернет-Университет Информационных Технологий (ИНТУИТ), 2016. — 147 с. — Режим доступа: http://www.iprbookshop.ru/52185.html

5. Сундукова Т.О. Структуры и алгоритмы компьютерной обработки данных [Электронный ресурс]/ Т.О. Сундукова, Г.В. Ваныкина. — Электрон. текстовые данные. — М.: Интернет-Университет Информационных Технологий (ИНТУИТ), 2016. — 749 с.— Режим доступа: http://www.iprbookshop.ru/57384.html

Электронные ресурсы:

6. Научная электронная библиотека http://elibrary.ru
7. Электронно-библиотечная система «ЛАНЬ» http://e.lanbook.com
8. Электронно-библиотечная система «IPRbooks» http://www.iprbookshop.ru
9. Электронно-библиотечная система «Юрайт» http://www.biblio-online.ru
10. Электронно-библиотечная система «Университетская библиотека ONLINE» http://www.biblioclub.ru