ЛАБОРАТОРНАЯ РАБОТА № 4
ЛИНЕЙНЫЕ СТРУКТУРЫ ДАННЫХ

Цель работы – познакомиться с динамическими структурами данных, научиться создавать абстрактные типы данных и решать задачи с использованием списков, стеков и очередей.

Теоретические сведения

Абстрактные типы данных и классы

Данные характеризуются тремя качествами: семантика, синтаксис и прагматика. ***Семантика*** – это содержательная характеристика данных, определяющая множество возможных значений данных и операций над ними без потери смысла. ***Синтаксис*** – это формализованная характеристика данных, определяющая внешний вид (структуру) данных. ***Прагматика*** – это характеристика, определяющая правила использования данных в ЭВМ, т.е. алгоритмы их обработки.

Комбинация семантической и синтаксической характеристик данных называют типом данных. Существует две группы типов данных – скалярные и структурированные.

Данные скалярных типов рассматриваются как целые простые неделимые объекты: int, float, double, char и т.д.

Данные структурированных типов представляются как составные агрегатные объекты, элементы которых могут быть данными одного и того же или различного типа. Значения структурного типа представляет собой совокупность значений

компонентов, относящихся к составляющим типам, и называются составными. Если для значений некоторого типа существует отношение порядка, то такой тип называется порядковым.

Определено несколько основных структур данных, на основе которых в программе могут строиться более сложные структуры, представляющие собой некоторые абстрактные данные (абстрактные типы данных).

Пару, содержащую элемент данных и элемент отношений, называют элементом структуры. Как и любые данные, структура данных представляется на трех уровнях. На внешнем уровне структура данных описывается как абстрактный объект. При этом определяются ее организация, возможные значения и операции, разрешенные над структурой, то есть определение структуры на внешнем уровне эквивалентно определению нового типа данных и его свойств. Отсюда можно определить, что ***абстрактный тип данных (АТД)*** – это математическая модель с совокупностью операторов, определенных в рамках этой модели. Простым примером АТД могут служить множества целых чисел с операторами объединения, пересечения и разности множеств. Абстрактный тип данных является независимым от реализации и позволяет программисту сосредоточиться на идеализированных моделях данных и операций над ними. Для описания АТД используется формат, который включает заголовок с именем АТД, описание типа данных и список операций. Для каждой операции определяются входные значения, предусловия, применяемые к

данным до того, как операция может быть выполнена, и процесс, который выполняется операцией. После выполнения операции определяются выходные значения и постусловия, указывающие на любые изменения данных. Большинство АТД имеют инициализирующую операцию, которая присваивает данным начальные значения.

```
АТД имя_типа
    Данные
        Описание структуры данных
    Операции
        Инициализатор
            Начальные значения:
                Данные, используемые для инициализации объекта
            Процесс:
                Инициализация объекта
        Операция1
            Вход:
                Входные данные
            Предусловия:
                Состояние системы перед выполнением операции
            Процесс:
                Действия, выполняемые с данными
            Выход:
                Данные, возвращаемые клиенту
            Постусловия:
```

Состояние системы после выполнения операций

Операция$_2$

…

Операция$_n$

Конец АТД

Для описания АТД на языке С++ лучше всего использовать классы.

Представление определенной структуры данных в памяти машины называется структурой хранения данных. Структура хранения, выбранная для представления конкретной структуры данных, должна, с одной стороны, сохранить отношения, существующие между элементами данных, с другой стороны, обеспечивать наиболее простое и эффективное выполнение операций над структурой данных в целом и ее отдельными элементами. Такими операциями являются создание и уничтожение структуры данных, включение и исключение ее элементов, доступ к элементам и т.д.

Наиболее простой структурой хранения многоэлементных структур данных является ***вектор***. Вектор размещается в одной сплошной области памяти. Элементы структуры данных в векторе физически размещаются последовательно один за другим. Поэтому в элементе структуры данных имеется только элемент данных и отсутствует элемент отношений, показывающий связи элемента структуры данных с другими элементами.

Такая структура хранения наиболее близко соответствует реальной организации памяти большинства ЭВМ. Все элементы структуры данных имеют одинаковую длину. Доступ к отдельному элементу осуществляется с использованием базового адреса (адреса начала) вектора и номера этого элемента (обычно в векторной памяти размещаются массивы, строки, стеки, очереди, таблицы).

Векторная структура хранения является ***полустатической***, то есть изменение длины структуры происходит в определенных пределах, не превышая какого-то максимального (предельного) значения.

Динамические структуры хранения характеризуются отсутствием физической смежности элементов структуры в памяти и непредсказуемостью числа элементов структуры в процессе ее обработки. 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: массивом, записью и т.п.;

2) поля связок, в котором содержатся один или несколько указателей, связывающих 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.

Достоинством связного представления данных является возможность 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.

Вместе с тем связное представление не лишено и недостатков:

- работа с указателями требует, как правило, более высокой квалификации от программиста;
- на поля связок расходуется дополнительная память;
- доступ к элементам связной структуры менее эффективен по времени в сравнении с вектором.

Последний недостаток является наиболее серьезным и именно им ограничивается применимость связного представления данных. Для связного представления адрес элемента не может быть вычислен. ***Дескриптор*** связной структуры содержит один или

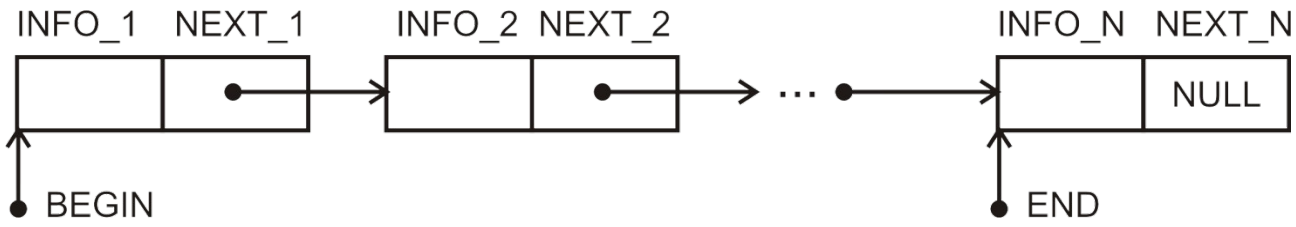
несколько указателей, позволяющих войти в структуру, далее поиск требуемого элемента выполняется следованием по цепочке указателей от элемента к элементу. Поэтому связное представление практически никогда не применяется в задачах, где логическая структура данных имеет вид вектора или массива – с доступом по номеру элемента, но часто применяется в задачах, где логическая структура требует другой исходной информации доступа (таблицы, списки, деревья и т.д.).

Список

Списком называется упорядоченное множество, состоящее из переменного числа элементов, к которым применимы операции включения, исключения и перебора элементов. Список, отражающий отношения соседства между элементами, называется линейным. Если ограничения на длину списка не допускаются, то список представляется в памяти в виде связной структуры.

Каждый элемент списка содержит по крайне мере два поля: одно поле содержит данные этого элемента или указатель на данные, а другое предназначено для размещения указателя на следующий элемент. Последний элемент такой структуры данных в поле указателя содержит признак конца списка (NULL). Различают однонаправленные, двунаправленные и циклические списки. Однонаправленный (односвязный) список является простейшей формой списка.

Структура однонаправленного линейного списка содержит



(рисунок 1): поле *INFO* – информационное поле, содержащее данные; *NEXT* – указатель на следующий элемент списка; *BEGIN* – указатель начала списка или «голова» списка; *END* – указатель окончания списка или «хвост» списка. В поле указателя последнего элемента списка находится специальный признак *NULL*, свидетельствующий о конце списка. Направление связей элементов в списке изображаются с помощью стрелок.

В памяти список представляет собой совокупность дескриптора и одинаковых по размеру и формату записей, размещенных произвольно в некоторой области памяти и связанных друг с другом в линейно упорядоченную цепочку с помощью указателей. Дескриптор списка реализуется в виде особой записи и содержит такую информацию о списке, как адрес начала списка, код структуры, имя списка, текущее число элементов в списке, описание элемента и т.д. Дескриптор может находиться в той же области памяти, в которой располагаются элементы списка, или память для него выделяется отдельно.

Ниже рассматриваются некоторые простые операции над линейными списками. К таким операциям относят: создание и

уничтожение списка, включение и исключение элемента из списка, поиск элемента в списке.

Выполнение операций включения и исключения элемента из односвязного линейного списка иллюстрируется в общем случае рисунками со схемами изменения связей (см. рисунки 2 и 3). На рисунках сплошными линиями показаны связи, имевшиеся до

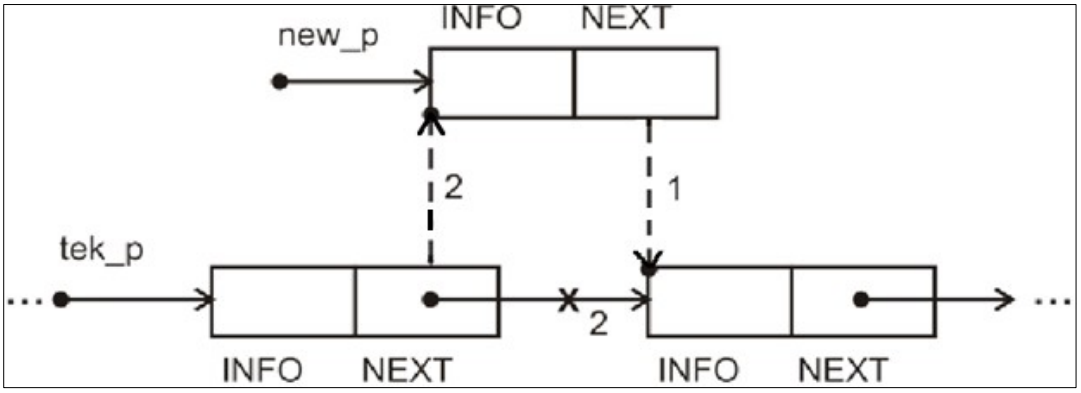


Рис. 2. Включение элемента в список

выполнения и сохранившиеся после выполнения операции. Пунктиром показаны связи, установленные при выполнении операции. Знаком «х» отмечены связи, разорванные при выполнении операции. Во всех операциях чрезвычайно важна последовательность изменения значений указателей, которая обеспечивает корректное изменение списка, не затрагивающее другие элементы (показана на рисунках с помощью цифр). При неправильном порядке выполнения действий легко потерять часть списка.

Для включения элемента в список необходимо проделать следующие действия, показанные на рисунке 2:

```
new_p->next = tek_p->next; /* присоединяем «хвост» списка к новому элементу */
```

```
tek_p->next = new_p; /* присоединяем к «голове» списка новый элемент */
```

Для исключения элемента из списка необходимо проделать следующие действия, показанные на рисунке 3:

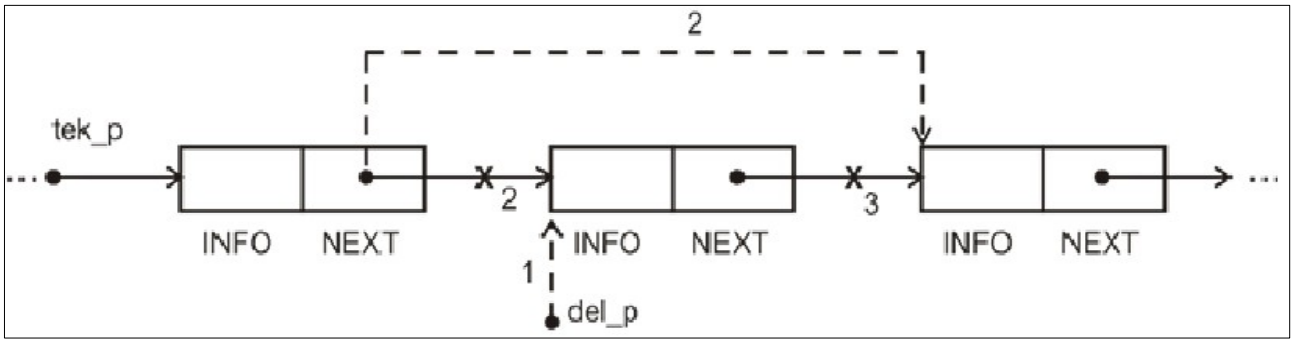


Рис. 3. Исключение элемента из списка

```
del_p = tek_p->next; /* запоминаем адрес удаляемого элемента */

tek_p->next = del_p->next; /* перенастраиваем связи элементов */

delete del_p; /* удаляем элемент */
```

Программный пример, полностью рассматривающий создание и уничтожение списка, включение и исключение элемента из списка, поиск элемента в списке, приведен в приложении 5.

Достоинствами односвязного списка являются простота создания, включения и исключения элементов, к недостаткам – невозможность просмотра элементов в противоположную сторону. Такую возможность обеспечивает двунаправленный (двусвязный)

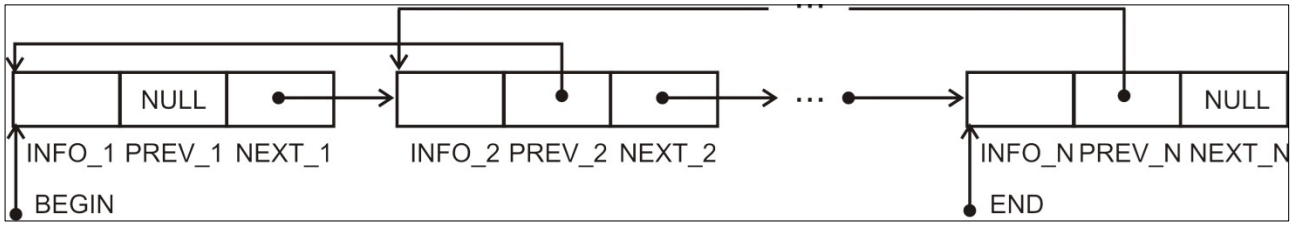


Рис. 4. Двунаправленный линейный списокРис 1. Однонаправленный линейный список

список, каждый элемент которого содержит два указателя: на следующий и предыдущий элементы списка.

Структура двунаправленного линейного списка содержит (рисунок 4): поле *INFO* – информационное поле, содержащее данные, *NEXT* – указатель на следующий элемент списка, поле *PREV* – указатель на предыдущий элемент списка, *BEGIN* – указатель начала списка или «голова» списка, *END* – указатель окончания списка или «хвост» списка. В крайних элементах соответствующие указатели содержат специальный признак *NULL*. Наличие двух указателей в каждом элементе усложняет список и приводит к дополнительным затратам памяти, но в то же время обеспечивает более эффективное выполнение некоторых операций над списком.

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 (рисунок 5).

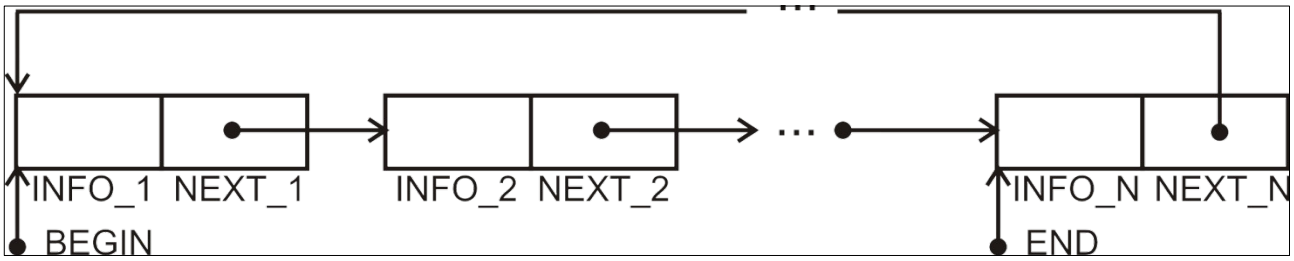


Рис. 5. Односвязный кольцевой список

При работе с такими списками несколько упрощаются некоторые процедуры, выполняемые над списком. Однако при просмотре такого списка следует принять меры предосторожности, чтобы не создать бесконечный цикл обработки данных.

Стек

Стек – это динамически изменяемый упорядоченный набор элементов, включение и исключение элементов из которого выполняются с одного и того же конца, называемого вершиной стека. Функционирование стека происходит по принципу ***LIFO*** (Last In First Out – последним пришел – первым исключается).

Основные операции над стеком – включение нового элемента (англ. *push* –заталкивать) и исключение элемента из стека (англ. *pop* – выскакивать). Полезными могут быть также вспомогательные операции: неразрушающее чтение элемента, находящегося на вершине стека, и очистка стека, которые могут быть реализованы комбинацией основных операций, проверка стека на пустоту и на полноту.

Стеки могут представляться в памяти в виде вектора или связного списка.

При векторном представлении под стек отводится сплошная область памяти, достаточно большая, чтобы в ней можно было поместить некоторое максимальное число элементов, которое определяется решаемой задачей. В дескрипторе этого вектора кроме обычных для вектора параметров должен находиться также указатель стека – индекс элемента, находящегося на вершине стека. При занесении элемента в стек сначала модифицируется указатель, а затем производится запись элемента. Модификация указателя состоит в прибавлении к нему или в вычитании из него единицы (в зависимости от того, растет стек в сторону увеличения или уменьшения адресов). Операция исключения элемента состоит в выборке значения, на которое указывает указатель стека и модификации указателя стека (в направлении, обратном модификации при включении). После выборки слот, в котором размещался выбранный элемент, считается свободным.

При реализации стека на основе связного списка и включение, и исключение элементов всегда производятся с начала списка. Дескриптор списка будет содержать один указатель – *top*, который всегда указывает на последний записанный в стек элемент. В исходном состоянии (при пустом стеке) значение указателя *top* равно NULL. При включении элемента для него выделяется память, а при исключении – освобождается. Перед включением элемента проверяется доступный объем памяти, и если он не позволяет выделить память для нового элемента, стек считается заполненным. При очистке стека последовательно уничтожаются его элементы.

Очередь

Очередью называется такой последовательный список с переменной длиной, в котором включение элементов выполняется только с одной стороны списка (хвоста очереди), а исключение – с другой стороны (из головы очереди). Часто для обозначения очереди используется аббревиатура ***FIFO*** (First In First Out – первым пришел – первым исключается).

Операции над очередью те же, что и над стеком: основные – включение и исключение, вспомогательные – неразрушающее чтение, очистка, проверка на пустоту, проверка на полноту.

Как и стеки, очереди могут представляться в памяти в виде вектора или связного списка.

При векторной реализации очереди в дополнение к обычным для дескриптора параметрам должны находиться два указателя (индекса): индекс начала очереди (первого элемента очереди) и индекс ее конца (первого свободного элемента в очереди). При включении элемента в очередь элемент записывается по адресу, определяемому указателем на конец, после чего этот указатель увеличивается на единицу. При исключении элемента из очереди выбирается элемент, адресуемый указателем на начало, после чего этот указатель также увеличивается на единицу.

Очевидно, что со временем указатель на хвост при очередном включении элемента достигнет верхней границы той области памяти, которая выделена для очереди. Однако если операции включения чередовались с операциями исключения элементов, то в начальной области отведенной под очередь памяти имеется свободное место. Для того чтобы память, занимаемая исключенными элементами, могла быть повторно использована, очередь замыкается в кольцо: указатели (на голову и на хвост), достигнув конца выделенной области памяти, переключаются на ее начало. Такая организация очереди в памяти называется кольцевой очередью.

В исходном состоянии указатели на голову и на хвост очереди указывают на начало области памяти. Равенство этих двух указателей (при любом их значении) является признаком пустой очереди. Если в процессе работы с кольцевой очередью число операций включения превышает число операций исключения, то может возникнуть ситуация, в которой указатель конца «догонит» указатель начала. Очередь будет заполненной, но если при этом указатели сравняются, понять, заполнена очередь или пуста, будет невозможно. Во избежание такой ситуации к кольцевой очереди предъявляется требование, чтобы между указателем на хвост и указателем на голову оставался «зазор» из свободных элементов. Когда этот «зазор» сокращается до одного элемента, очередь считается заполненной, и дальнейшие попытки записи в нее блокируются. Очистка очереди сводится к записи одного и того же (необязательно начального) значения в оба указателя. Пример организации кольцевой очереди приведен в приложении 6.

При реализации очереди на основе связного списка включение элементов производится в конец списка, исключение элементов – с начала списка. Дескриптор списка будет содержать два указателя, один – на первый, второй – на последний записанный элемент. В исходном состоянии (при пустой очереди) значение обоих указателей равно NULL. При включении элемента для него выделяется память, а при исключении – освобождается. Перед включением элемента проверяется доступный объем памяти, и если он не позволяет выделить память для нового элемента, очередь считается заполненной. При очистке очереди последовательно уничтожаются его элементы.

Дек

Дек (англ. deque – double ended queue, т.е. очередь с двумя концами) – это такой последовательный список, в котором как включение, так и исключение элементов может осуществляться с любого из двух концов списка. Однако, применительно к деку целесообразно говорить не о начале и

конце, а о левом и правом концах списка. Операции над деком: включение элемента справа, включение элемента слева, исключение элемента справа, исключение элемента слева.

Как и стеки, очереди могут представляться в памяти в виде вектора или связного списка.

Физическая структура дека в векторном представлении идентична структуре кольцевой очереди. Единственное отличие – изменение индексов левого и правого концов дека возможно не только в сторону увеличения, но и в сторону уменьшения адресов. Естественно, что при достижении нижней границы вектора указатель должен быть переставлен на его верхнюю границу.

При организации дека в списковой структуре нужно учитывать, что при удалении последнего элемента односвязного списка нужно обнулить ссылочное поле предшествующего элемента. В односвязном списке это возможно сделать, только перебрав все элементы списка от первого до предпоследнего. Поэтому при необходимости использования именно связанной структуры хранения дек реализуется на базе двусвязного списка.

Частный случай дека – дек с ограниченным входом и дек с ограниченным выходом. В первом случае добавление элемента возможно только в один конец списка, а удаление – с обоих. Во втором добавление элементов осуществляется в оба конца списка, а удаление – только с одного. Соответственно при списковой реализации дек с ограниченным входом потребует наличия двух связей, а для дека с ограниченным выходом будет достаточно односвязного списка.

Постановка задачи

Написать две программы согласно номеру индивидуального варианта. В первом задании данные хранятся в бинарном файле

записей (можно использовать файл, созданный при выполнении лабораторной работы №1), а для обработки считываются в список. При выходе из программы обработанные данные сохраняются в том же файле. Список должен описываться классом. Для организации интерфейса в программе должно использоваться меню. Во втором задании необходимо создать класс, описывающий стек или очередь, и программу решения поставленной задачи с использованием объекта этого класса.

Задания могут быть выполнены на трех уровнях сложности.

1) Низкий. В первом задании список линейный односвязный. Обязательные методы класса: добавление элемента в начало или конец списка (на выбор), просмотр списка, удаление элемента из начала или конца списка (на выбор). Вывод списка на экран можно выполнять в любом (главное, читабельном) виде. Во втором задании реализовать определенную вариантом структуру данных (стек или очередь) любым удобным способом, вместо решения поставленной задачи можно написать программу тестирования объекта созданного класса.

2) Средний. В первом задании список линейный односвязный. Обязательные методы класса: добавление элемента в упорядоченный список с сохранением упорядоченности (ключевое поле выбрать самостоятельно), просмотр списка, удаление произвольного элемента списка. Вывод данных осуществлять в табличном виде с графлением подходящими символами. Во втором задании создать класс, описывающий требуемую структуру данных, в соответствии с заданным вариантом реализации.

3) Высокий. В первом задании список линейный двусвязный. Обязательные методы класса: добавление элемента в произвольное место списка, просмотр списка в прямом и обратном направлении, удаление произвольного элемента списка. Вывод данных осуществлять постранично в

табличном виде с графлением подходящими символами. Во втором задании создать необходимую для решения задачи структуру данных с помощью двух структур хранения: векторной и списковой,– реализацию оформить в виде классов с единым интерфейсом.

Варианты реализаций

Реализация 1. Стек в массиве. Заполнение стека должно производиться с начала массива. Методы класса: добавление элемента в стек, удаление элемента из стека, получение значения с вершины стека, проверка заполнения стека, проверка пустоты стека.

Реализация 2. Разработайте класс, реализующий стек с помощью указателей. Методы класса: добавление элемента в стек, удаление элемента из стека, получение значения с вершины стека, проверка заполнения стека, проверка пустоты стека, очистка стека.

Реализация 3. Разработайте класс, реализующий очередь в «циклическом» массиве. Поля класса: массив, индексы первого и последнего элементов в очереди. Методы класса: добавление элемента в очередь, удаление элемента из очереди, получение значения из очереди, проверка заполнения очереди, проверка пустоты очереди.

Реализация 4. Разработайте класс, реализующий очередь в «циклическом» массиве. Поля класса: массив, индекс первого элемента в очереди, количество элементов в очереди. Методы класса: добавление элемента в очередь, удаление элемента из очереди, получение значения из очереди, проверка заполнения очереди, проверка пустоты очереди.

Реализация 5. Разработайте класс, реализующий очередь с помощью указателей. Методы класса: добавление элемента в очередь, удаление элемента из очереди, получение значения из очереди, проверка заполнения очереди, проверка пустоты очереди.

Контрольные вопросы

1. Что такое абстрактный тип данных?
2. Что такое список?
3. Что такое связанный список?
4. Какие виды списков Вы знаете?
5. Какие методы применимы к спискам?
6. Какие методы реализации списков Вы знаете?
7. Что такое стек?
8. Какие операции применимы к стекам?
9. Каков механизм заполнения стека? Что такое «дно» стека?
10. Что такое очередь?
11. Какие операции применимы к очередям?
12. Каков механизм заполнения очереди?
13. Что такое дек? Какие операции применимы к декам?
14. В чем различие между конкатенацией двух стеков и конкатенацией двух очередей?
15. Что такое дескриптор списка?
16. Как и когда используется нулевой указатель *NULL*?
17. Какая структура данных описывается аббревиатурой LIFO?
18. Какая структура данных описывается аббревиатурой FIFO?

19. Что такое «структура данных» и «структура хранения»?

20. От чего зависит выбор структуры хранения для реализации структуры данных?

Варианты заданий

Вариант 17

1. Поля данных: фамилия, баллы по математике, русскому и английскому языкам. Известны проходная сумма баллов и минимальное допустимое количество баллов по каждой дисциплине. Вывести список абитуриентов, имеющих наибольшую сумму баллов, и процент абитуриентов, не выдержавших конкурса.

2.Написать программу, использующую класс (реализация 2) для отыскания прохода по лабиринту. Лабиринт представляется в виде матрицы, состоящей из квадратов. Каждый квадрат либо открыт, либо закрыт. Вход в закрытый квадрат запрещен. Если квадрат открыт, то вход в него возможен со стороны, но не с угла. Каждый квадрат определяется его координатами в матрице. После отыскания прохода программа печатает найденный путь в виде координат квадратов.