Лабораторная работа №5

«Структуры данных Очереди»

Задание 1.

Создать класс **(не использовать шаблоны STL, boost)**, реализующий методы работы с очередью. Написать программу, иллюстрирующую работу всех методов работы с очередью. Результат формирования и преобразования очереди отображать в компонентах ListBox или его аналогах. После этого на базе родительского класса написать свой класс, реализующий метод решения своего варианта. Написать обработчик события, реализующий вызов метода решения своего варианта. Структуры данных **необходимо** реализовать на основе “Умных указателей с++”.

Индивидуальные задания

1. Создать очередь из случайных целых чисел. Найти минимальный элемент и сделать его первым.
2. Создать две очереди из случайных целых чисел. В первой найти максимальный элемент и за ним вставить элементы второй очереди.
3. Создать двухсвязанный список из случайных целых чисел. Удалить из списка все элементы, находящиеся между максимальным и минимальным.
4. Упорядочить элементы двухсвязанного списка случайных целых чисел в порядке возрастания методом «пузырька», когда можно переставлять местами только два соседних элемента.
5. Представить текст программы в виде двухсвязанного списка. Задать номера начальной и конечной строк. Этот блок строк следует переместить в заданное место списка.
6. Создать двухсвязанный список из случайных целых чисел. Удалить все отрицательные элементы списка.
7. Создать двухсвязанный список из случайных целых чисел. Из элементов, расположенных между максимальным и минимальным, создать первое кольцо. Остальные элементы должны составить второе кольцо.
8. Создать двухсвязанный список из случайных целых, положительных и отрицательных чисел. Из этого списка образовать два списка, первый из которых должен содержать отрицательные числа, а второй – положительные. Элементы списков не должны перемещаться в памяти.
9. Создать двухсвязанный список из строк программы. Преобразовать его в кольцо. Организовать видимую в компоненте TMemo или его аналоге циклическую прокрутку текста программы.
10. Создать два двухсвязанных списка из случайных целых чисел. Вместо элементов первого списка, заключенных между максимальным и минимальным, вставить второй список.
11. Создать двухсвязанный список из случайных целых чисел. Удалить из списка элементы с повторяющимися более одного раза значениями.

12. Создать двухсвязанный список и поменять в нем элементы с максимальным и минимальным значениями, при этом элементы не должны перемещаться в памяти.
13. Создать двухсвязанный список из нарисованных вами картинок. Преобразовать его в кольцо и организовать его циклический просмотр в компоненте TImage или его аналоге.
14. Создать двухсвязанный список из случайных чисел. Преобразовать его в кольцо. Предусмотреть возможность движения по кольцу в обе стороны с отображением места положения текущего элемента с помощью компоненты TGauge(Kind=gkPie) или их аналогах и числового значения – с помощью TLabel или его аналога.
15. Создать двухсвязанный список из текста вашей программы и отобразить его в TListBox или его аналоге. Выделить в TListBox или его аналоге часть строк и обеспечить запоминание этих строк. Далее выделить любую строку и нажать кнопку, которая должна обеспечивать перемещения выделенных ранее строк перед текущей строкой. При этом в TListBox или его аналоге должны отображаться строки из двухсвязанного списка.

Задание 2. Deque

Deque (double-ended queue) - индексируемая двусвязная очередь, поддерживающее следующие операции, каждое из которых работает за константу:

• *push_back(x)* - добавляет *x* в конец очереди.

• *push_front(x)* - добавляет *x* в начало очереди.

• *pop_back()* - удаляет последний элемент из очереди.

•*pop_front()* - удаляет первый элемент из очереди.

• *random access* индексирование.

Простейший *deque<T>* (здесь и далее за *T* будем считать тип данных с которым позволяет работать контейнер deque) представляет из себя некоторый массив типа *T* размера *capacity*, из которых задействовано лишь *size* элементов (размер *deque*).

При добавлении нового элемента в *deque* мы обращаемся к зарезервированным и еще не используемым элементам массива и, при отсутствии таковых, создаем новый массив размера $capacity*k$, после чего можно переместить значения старого массива в новый и, наконец, добавить новый элемент. Такой подход используется и в *vector*, что позволяет обходиться без больших расходов на память.

Итераторы могут инвалидироваться, храня указатели на элементы старого массива. Предложенный далее алгоритм позволит *resize deque* без инвалидации итераторов. Немного о процессе добавления/удаления элементов из обычного *deque*.

В любой момент времени необходимо поддерживать два, скажем так, указателя:

• первый (левый) указывает на начало очереди

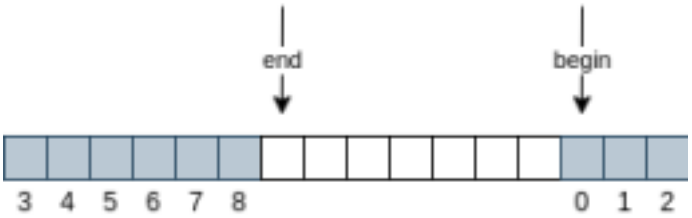
• второй (правый) указывает на следующий элемент после последнего в очереди

Для того чтоб добавить элемент в конец очереди достаточно положить его в ячейку массива, на которую указывает второй указатель, после чего инкрементировать его.

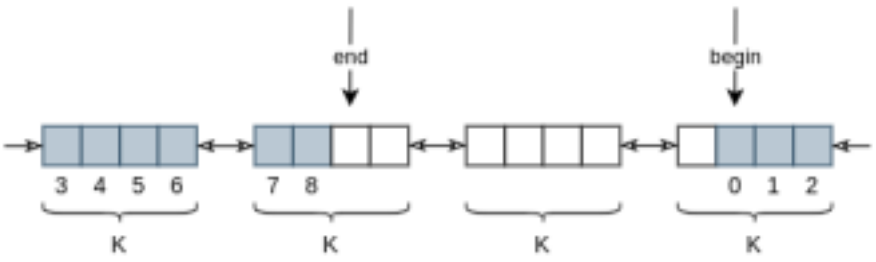
Для добавления элемента в начало очереди необходимо декрементировать левый указатель и положить туда добавляемый элемент. Делаем каждый раз *resize* при добавлении, когда *capacity==size* и по лучаем амортизированную сложность O(1).

Удаление - действия, обратные добавлению.

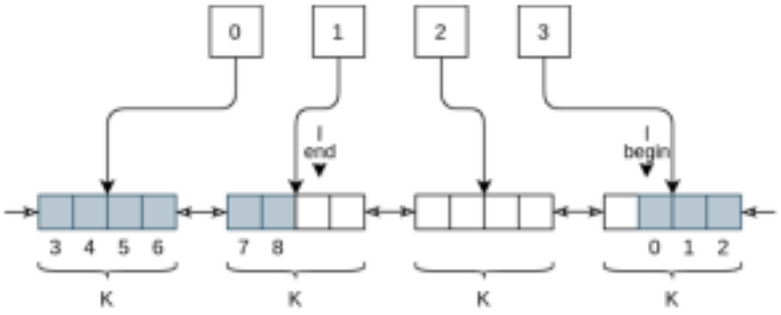
При инеркментировании/декрементировании указателей следует отме тить, что первый и последний элементы выделенного массива связаны (т.е. после инкрементирования указателя на последний элемент массива он должен указывать на первый).



Чтоб получить структуру, где итераторы не инвалидируются разобьём выделенный массив на блоки фиксированного размера *K*.

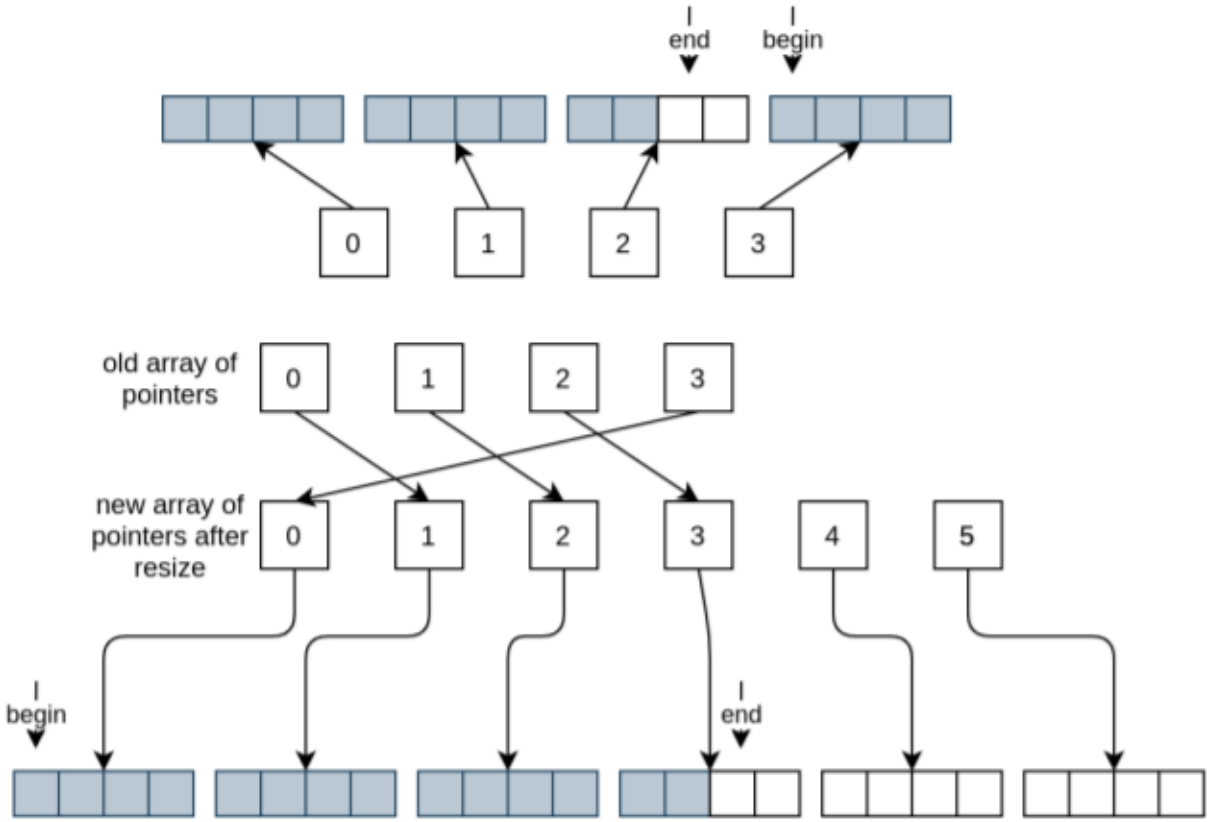


Для возможности *random access* индексирования создадим массив размера cap/k (*capacity* делится на *K*), который будет хранить указатели на блоки.



При попытке инкрементировать указатель (на объект), указывающий на последний элемент блока, то указатель в кольце из блоков перемещается на первый элемент следующего блока. Аналогично и при декременте. *resize* теперь будет проводиться над массивом указателей.

Может быть ситуация, когда при добавлении необходимо сдвинуть указатель в соседний блок со свободными элементами, однако писать туда будет нельзя, так как при *resize* тот блок, в котором находится первый элемент очереди должен быть под первым указателем нового массива. Иными словами, начальный блок должен оказаться в начале, а конечный в конце. Но если в одном блоке получится, что *end* левее *begin*, то этот блок придется разбить, что тоже приведет к инвалидации итераторов. Потому следует наложить запрет на то, чтоб в одном блоке *end* был левее *begin*.



Задача:

Реализовать описанную структуру данных со следующими свойствами и продемонстрировать её работу при помощи визуальных компонентов:

• *push_back* - O(1)

• *push_front* - O(1)

• *pop_back* - O(1)

• *pop_front* - O(1)

• *clear*

• *size*

• *empty*

• *random access iterator* без инвалидации при *resize*, который тоже надо будет реализовать.