Бригадные задания

Вариант	Задание
1	Разработать класс, реализующий бинарное дерево. У данного класса должны быть предусмотрены методы: а) `insert()` — добавление элемента в дерево; б) `find()` — найти элемент в дереве (возвращает **`true`**, если такой элемент в дереве имеется и **`false`**, если искомого элемента нет); в) `delete()` — удаляет элемент из дерева, если такого элемента в дереве нет, то ничего не делает; г) `balance()` — производит балансировку дерева путём полного перестроения дерева; д) `print()` — вывод всех элементов дерева на печать. *Примечание.* Разумно организовать дерево следующим образом: хранится адрес узла-вершины дерева, а каждый узел хранит адрес родительского узла, адреса левого и правого поддерева, а также хранимое значение.
2	Написать две функции: а) `DoublyLinkedList toDoublyLinkedList(const SinglyLinkedList& source)` — производит преобразование односвязного списка в двусвязный; б) `SinglyLinkedList toSinglyLinkedList(const DoublyLinkedList& source)` — производит преобразование двусвязного списка в односвязный.
3	Разработать класс, реализующий красно-черное дерево. У данного класса должны быть предусмотрены методы: а) `insert()` — добавление элемента в дерево; б) `find()` — найти элемент в дереве (возвращает **`true`**, если такой элемент в дереве имеется и **`false`**, если искомого элемента нет); в) `delete()` — удаляет элемент из дерева, если такого элемента в дереве нет, то ничего не делает; г) `balance()` — производит балансировку дерева путём полного перестроения дерева;

д) `print()` — вывод всех элементов дерева на печать.

Примечание. Разумно организовать дерево следующим образом: храниться адрес вершины дерева, каждый узел хранит адрес родительского узла, адреса левого и правого поддерева, хранимое значение и цвет узла.

4 Добавить в класс односвязного списка `SinglyLinkedList` статические методы сохранения в бинарный файл и загрузки списка из него:

а) `void SaveToFile(const SinglyLinkedList& source, const char* filename)`, который выполняет сохранение списка `source` в бинарный файл с именем `filename`;

б) `SinglyLinkedList LoadFromFile(const char* filename)`, который возвращает односвязный список, загруженный из бинарного файла с именем `filename`.

5 Добавить в класс односвязного списка `DoublyLinkedList` статические методы сохранения в бинарный файл и загрузки списка из него:

а) `void SaveToFile(const DoublyLinkedList& source, const char* filename)`, который выполняет сохранение списка `source` в бинарный файл с именем `filename`;

б) `DoublyLinkedList LoadFromFile(const char* filename)`, который возвращает односвязный список, загруженный из бинарного файла с именем `filename`.

6 Реализовать класс хэш-таблицы `HashTable`, хранящей пары (ключ, значение) типа `(size_t, const char*)`. Наибольшее возможное количество элементов N задается параметром конструктора и не меняется. Методы `HashTable`:

а) `void add(const size_t key, const char* value)`, добавляющий в таблицу значение `value` с ключом `key`;

б) `void remove(const size_t key)`, удаляющий из таблицы пару с ключом `key`;

в) `const char* find(const size_t key)`, возвращающий значение в паре с ключом `key`, или нулевой указатель, если такой пары нет в таблице.

г) `size_t getTotalCount() const`, возвращающий общее число пар в таблице.

Указание. В качестве хэш-функции можно использовать остаток от деления на N. Ячейки таблицы предлагается представить массивом указателей на односвязные списки, тогда коллизии можно разрешать, добавляя элементы с одинаковыми значениями хэш-функции в список одной ячейки.

7 Реализовать функцию, выполняющую упорядочивание элементов коллекции методом пузырьковой сортировки. Функция принимает в качестве аргумента ссылку на объект класса, реализующего интерфейс `ICollection`.

8 Реализовать функцию, выполняющую упорядочивание элементов коллекции методом сортировки Шелла. Функция принимает в качестве аргумента ссылку на объект класса, реализующего интерфейс `ICollection`.

9 Разработать функцию, которая принимает в качестве аргумента ссылку на связный список и разворачивает его. Разворот должен осуществляться с помощью разработанного ранее класса очереди.

10 Реализовать класс кольцевого буфера `CircularBuffer` с интерфейсом `ICollection`. Началом и концом кольцевого буфера является один и тот же элемент: первый добавленный в буфер, а при его удалении таковым становится второй. В класс необходимо добавить метод `Iterator getIterator() const`, где `Iterator` — это класс, позволяющий выполнять обход кольцевого буфера ровно один раз от начала:

```
class Iterator {
public:
	bool moveNext();
	Type getCurrent() const;
};
```

Метод `moveNext()` возвращает `true`, если итератор еще не обошел кольцевой буфер и успешно перешел к следующему элементу, иначе метод возвращает **`false`**. Метод `getCurrent()` возвращает значение элемента буфера, на котором в данный момент находится итератор. Изначально итератор не находится ни на каком элементе, то есть, после первого вызова `moveNext()` с результатом **`true`** метод `getCurrent()` возвращает значение первого элемента в буфере.

11 Написать функцию, которая принимает в качестве аргумента массив указателей на объекты, реализующие интерфейс `ICollection`, и возвращающую односвязный список, содержащий все элементы всех переданных в функцию коллекций. Элементы в возвращаемом списке упорядочены по возрастанию.

12 Написать функцию, которая принимает в качестве аргумента массив указателей на объекты, реализующие интерфейс `ICollection`, и возвращающую двусвязный список, содержащий все элементы всех переданных в функцию коллекций. Элементы в возвращаемом списке упорядочены по возрастанию.

13 Реализовать очередь с приоритетами. При добавлении элемента в очередь необходимо также указать его приоритет. Возможны три вида приоритетов элементов: низкий, обычный, высокий. Порядок извлечения элементов из очереди:

а) элементы с высоким приоритетом, но не более 4-х подряд;

б) элементы с обычным приоритетом, но не более 3-х подряд;

в) элементы с низким приоритетом.

Пример: если в очереди 5 элементов с высоким приоритетом, 4 — с обычным и 2 — с низким, то порядок извлечения: 4 элемента с высоким, 3 — с обычным, 1 — с низким, 1 — с высоким, 1 — с обычным и 1 элемент с низким приоритетом.

Обработку ошибок следует реализовать при помощи исключений. В качестве класса для объекта-исключения подойдут `std::logic_error` и `std::out_of_range` из заголовочного файла `<stdexcept>`. Допускается использовать и собственные классы-исключения.