Задание 1 для первого семестра 2 курса

Это примерный список задач для первого задания. Все задачи так или иначе требуют реализации списочных структур. Первые задачи на списки и массивы совсем простые. Последующие требуют некоторых размышлений. На выполнение этих заданий можно отвести месяц – полтора, после чего зафиксировать факт выполнения или невыполнения задания.

Второе задание будет связано с реализацией древовидных структур, хеш-множеств и других задач, близких к материалу лекций.

Реализацию следует выполнять на C++. Задачи экзамена также потребуют знания основ C++. Так как на лекциях не будет последовательного изложения языка C++, этому следует посвятить некоторое время на семинарских занятиях.

Требования к реализации

0. Если кто еще не умеет, то должен научиться писать реализацию в нескольких файлах, использовать собственные заголовочные файлы, уметь составлять простейшие make-файлы и использовать их для сборки, а также уметь собирать программу из командной строки.

1. Все реализации должны удовлетворять описанным неформальным интерфейсам и иметь заголовочный .h файл с прокоментированным описанием методов класса. Разработка формального описания классов предоставляется студенту. при этом должны быть предусмотрены способы реакции и оповещения о некорректных или ошибочных ситуациях.

2. К заданию должны быть написаны тесты, т.е. отдельные программы которая вызывает требуемые функции с разнообразными наборами данных и (желательно автоматически) проверяет правильность результата. Тесты должны быть жесткими, т.е. направленными на то, чтобы сломать реализацию (большие массивы, длинные строки, некорректные вызовы и пр.) В некоторых задачах тесты должны моделировать различные сценарии использования классов (например, несколько раз добавили, несколько раз забрали элементы; и так в цикле, скажем, 1000000 раз). В задачах, имеющих дело со строками, в тестах следует предусмотреть возможность получения строк/слов для теста чтением из текстовых файлов.

3. В рамках тестов можно предложить добавить методы, подсчитывающие вычислительные и временны́е затраты. Например, количество времени на выполнение тех или иных действий.

4. При приеме задания желательно проверить умение студента написать какой-либо новый тест к своей реализации тут же на семинаре без посторонней помощи.

Задание 1.1. *Список строк с возможностью сортировки.*

Требуется реализовать список строк (указателей `char *`) с возможностью его сортировки по разным критериям. Предполагается, что после добавления строки в список ее длина измениться не может. Поэтому для размещения строки в памяти используется new (или malloc), а полученные указатели сохраняются в списке.

Реализация класса должна обеспечивать следующие возможности:

— создать пустой список;
— добавить элемент в конец списка;
— добавить в конец все элементы другого списка;
— вставить элемент так, чтобы он имел указанный индекс (порядковый номер);
— встать в конец / начало списка;
— получить указатель на следующий / предыдущий элементы списка (перемещение текущей позиции и итератор);
— получить индекс (порядковый номер) текущего элемента;
— получить указатель (или ссылку) текущего элемента для непосредственного доступа к значению;
— удалить текущий элемент или элемент по указанному индексу;
— получить количество элементов в списке;
— удалить все элементы из списка;
— сортировать список по указанной функции сравнения строк.

Формальное определение интерфейса не задается и должно быть разработано студентом.

Тесты должны включать загрузку списка строками из файла и проверку работы всех реализованных методов в различных корректных и некорректных ситуациях.

В качестве метода сортировки следует использовать алгоритм выбора и перестановки максимального элемента.

Задание 1.2. *Список строк с возможностью сортировки.*

Требуется реализовать список строк (указателей `char *`) с возможностью его сортировки по разным критериям. Предполагается, что после добавления строки в список ее длина измениться не может. Поэтому для размещения строки в памяти используется new (или malloc), а полученные указатели сохраняются в списке.

Реализация класса должна обеспечивать следующие возможности:

— создать пустой список;
— добавить элемент в конец списка;
— добавить в конец все элементы другого списка;
— вставить элемент так, чтобы он имел указанный индекс (порядковый номер);
— встать в конец / начало списка;
— получить указатель на следующий / предыдущий элементы списка (перемещение текущей позиции и итератор);
— получить индекс (порядковый номер) текущего элемента;
— получить указатель (или ссылку) текущего элемента для непосредственного доступа к значению;
— удалить текущий элемент или элемент по указанному индексу;
— получить количество элементов в списке;
— удалить все элементы из списка;
— сортировать список по указанной функции сравнения строк.

Формальное определение интерфейса не задается и должно быть разработано студентом.

Тесты должны включать загрузку списка строками из файла и проверку работы всех реализованных методов в различных корректных и некорректных ситуациях.

В качестве метода сортировки следует использовать алгоритм пузырьковой сортировки.

Задание 1.3. *Список строк с возможностью сортировки.*

Требуется реализовать список строк (указателей `char *`) с возможностью его сортировки по разным критериям. Предполагается, что после добавления строки в список ее длина измениться не может. Поэтому для размещения строки в памяти используется new (или malloc), а полученные указатели сохраняются в списке.

Реализация класса должна обеспечивать следующие возможности:

— создать пустой список;
— добавить элемент в конец списка;
— добавить в конец все элементы другого списка;
— вставить элемент так, чтобы он имел указанный индекс (порядковый номер);
— встать в конец / начало списка;
— получить указатель на следующий / предыдущий элементы списка (перемещение текущей позиции и итератор);
— получить индекс (порядковый номер) текущего элемента;
— получить указатель (или ссылку) текущего элемента для непосредственного доступа к значению;
— удалить текущий элемент или элемент по указанному индексу;
— получить количество элементов в списке;
— удалить все элементы из списка;
— сортировать список по указанной функции сравнения строк.

Формальное определение интерфейса не задается и должно быть разработано студентом.

Тесты должны включать загрузку списка строками из файла и проверку работы всех реализованных методов в различных корректных и некорректных ситуациях.

В качестве метода сортировки следует использовать алгоритм просеивания (модифицированный пузырек).

Задание 1.4. *Список строк с возможностью сортировки.*

Требуется реализовать список строк (указателей `char *`) с возможностью его сортировки по разным критериям. Предполагается, что после добавления строки в список ее длина измениться не может. Поэтому для размещения строки в памяти используется new (или malloc), а полученные указатели сохраняются в списке.

Реализация класса должна обеспечивать следующие возможности:

— создать пустой список;
— добавить элемент в конец списка;
— добавить в конец все элементы другого списка;
— вставить элемент так, чтобы он имел указанный индекс (порядковый номер);
— встать в конец / начало списка;
— получить указатель на следующий / предыдущий элементы списка (перемещение текущей позиции и итератор);
— получить индекс (порядковый номер) текущего элемента;
— получить указатель (или ссылку) текущего элемента для непосредственного доступа к значению;
— удалить текущий элемент или элемент по указанному индексу;
— получить количество элементов в списке;
— удалить все элементы из списка;
— сортировать список по указанной функции сравнения строк.

Формальное определение интерфейса не задается и должно быть разработано студентом.

Тесты должны включать загрузку списка строками из файла и проверку работы всех реализованных методов в различных корректных и некорректных ситуациях.

В качестве метода сортировки следует использовать алгоритм слияния упорядоченных списков.

Задание 1.5. *Список строк с возможностью сортировки.*

Требуется реализовать список строк (указателей `char *`) с возможностью его сортировки по разным критериям. Предполагается, что после добавления строки в список ее длина измениться не может. Поэтому для размещения строки в памяти используется new (или malloc), а полученные указатели сохраняются в списке.

Реализация класса должна обеспечивать следующие возможности:

— создать пустой список;
— добавить элемент в конец списка;
— добавить в конец все элементы другого списка;
— вставить элемент так, чтобы он имел указанный индекс (порядковый номер);
— встать в конец / начало списка;
— получить указатель на следующий / предыдущий элементы списка (перемещение текущей позиции и итератор);
— получить индекс (порядковый номер) текущего элемента;
— получить указатель (или ссылку) текущего элемента для непосредственного доступа к значению;
— удалить текущий элемент или элемент по указанному индексу;
— получить количество элементов в списке;
— удалить все элементы из списка;
— сортировать список по указанной функции сравнения строк.

Формальное определение интерфейса не задается и должно быть разработано студентом.

Тесты должны включать загрузку списка строками из файла и проверку работы всех реализованных методов в различных корректных и некорректных ситуациях.

В качестве метода сортировки следует использовать алгоритм быстрой сортировки.

Задание 1.6. *Динамический массив чисел с возможностью сортировки.*

Требуется реализовать динамический массив чисел (значений `double`) с возможностью сортировки и быстрого (бинарного) поиска. Идея реализации состоит в том, что сначала для хранения выделяется небольшой массив фиксированной длины, а по мере добавления элементов выделяются дополнительные такие же массивы, которые связываются в список.

Реализация класса должна обеспечивать следующие возможности:

— создать массив заданной начальной длины;
— добавить элемент в конец массива (удлиннить массив);
— вставить элемент так, чтобы он имел указанный индекс (удлиннить массив);
— получить указатель (или ссылку) элемента с заданным индексом для прямого доступа к значению;
— удалить элемент по указанному индексу (сократить массив);
— получить количество элементов в массиве;
— установить новую длину массива;
— искать заданное значение (с заданным допуском) в массиве;
— сортировать массив по возрастанию или убыванию.

Формальное определение интерфейса не задается и должно быть разработано студентом.

Тесты должны включать заполнение массива некоторыми значениями и проверку работы всех реализованных методов в различных корректных и некорректных ситуациях.

В качестве метода сортировки используется алгоритм выбора и перестановки максимального элемента.

Задание 1.7. *Динамический массив чисел с возможностью сортировки.*

Требуется реализовать динамический массив чисел (значений `double`) с возможностью сортировки и быстрого (бинарного) поиска. Идея реализации состоит в том, что сначала для хранения выделяется небольшой массив фиксированной длины, а по мере добавления элементов выделяются дополнительные такие же массивы, которые связываются в список.

Реализация класса должна обеспечивать следующие возможности:

— создать массив заданной начальной длины;
— добавить элемент в конец массива (удлиннить массив);
— вставить элемент так, чтобы он имел указанный индекс (удлиннить массив);
— получить указатель (или ссылку) элемента с заданным индексом для прямого доступа к значению;
— удалить элемент по указанному индексу (сократить массив);
— получить количество элементов в массиве;
— установить новую длину массива;
— искать заданное значение (с заданным допуском) в массиве;
— сортировать массив по возрастанию или убыванию.

Формальное определение интерфейса не задается и должно быть разработано студентом.

Тесты должны включать заполнение массива некоторыми значениями и проверку работы всех реализованных методов в различных корректных и некорректных ситуациях.

В качестве метода сортировки используется алгоритм пузырьковой сортировки.

Задание 1.8. *Динамический массив чисел с возможностью сортировки.*

Требуется реализовать динамический массив чисел (значений `double`) с возможностью сортировки и быстрого (бинарного) поиска. Идея реализации состоит в том, что сначала для хранения выделяется небольшой массив фиксированной длины, а по мере добавления элементов выделяются дополнительные такие же массивы, которые связываются в список.

Реализация класса должна обеспечивать следующие возможности:

— создать массив заданной начальной длины;
— добавить элемент в конец массива (удлиннить массив);
— вставить элемент так, чтобы он имел указанный индекс (удлиннить массив);
— получить указатель (или ссылку) элемента с заданным индексом для прямого доступа к значению;
— удалить элемент по указанному индексу (сократить массив);
— получить количество элементов в массиве;
— установить новую длину массива;
— искать заданное значение (с заданным допуском) в массиве;
— сортировать массив по возрастанию или убыванию.

Формальное определение интерфейса не задается и должно быть разработано студентом.

Тесты должны включать заполнение массива некоторыми значениями и проверку работы всех реализованных методов в различных корректных и некорректных ситуациях.

В качестве метода сортировки используется алгоритм просеивания (модифицированный пузырек).

Задание 1.9. *Динамический массив чисел с возможностью сортировки.*

Требуется реализовать динамический массив чисел (значений `double`) с возможностью сортировки и быстрого (бинарного) поиска. Идея реализации состоит в том, что сначала для хранения выделяется небольшой массив фиксированной длины, а по мере добавления элементов выделяются дополнительные такие же массивы, которые связываются в список.

Реализация класса должна обеспечивать следующие возможности:

— создать массив заданной начальной длины;
— добавить элемент в конец массива (удлиннить массив);
— вставить элемент так, чтобы он имел указанный индекс (удлиннить массив);
— получить указатель (или ссылку) элемента с заданным индексом для прямого доступа к значению;
— удалить элемент по указанному индексу (сократить массив);
— получить количество элементов в массиве;
— установить новую длину массива;
— искать заданное значение (с заданным допуском) в массиве;
— сортировать массив по возрастанию или убыванию.

Формальное определение интерфейса не задается и должно быть разработано студентом.

Тесты должны включать заполнение массива некоторыми значениями и проверку работы всех реализованных методов в различных корректных и некорректных ситуациях.

В качестве метода сортировки используется алгоритм быстрой сортировки.

Задание 1.10. *Битовое множество.*

Требуется реализовать хранение и работу с подмножествами целых чисел, принадлежащих заданному диапазону. Идея реализации: заводится массив целых чисел, в котором каждому возможному элементу множества сопоставляется один бит. Таким образом, работа сводится к установке или проверке значения отдельного бита в некотором элементе целочисленного массива. Значения, выходящие из заданного диапазона, не могут размещаться в множестве (сообщение об ошибке), но диапазон можно изменить отдельной функией.

Реализация класса должна поддерживать следующие возможности:

— создание пустого множества для заданного диапазона чисел;
— очистка всего множества;
— добавление / удаление заданного значения;
— проверка принадлежности заданного значения множеству;
— получение максимального и минимального значений, содержащихся в множестве;
— итератор по множеству (возможность последовательного перебора всех значений из множества);
— объединение, пересечение и другие операции с аналогичным множеством;
— изменение диапазона допустимых элементов;
— получение границ текущего диапазона множества.

Формальное определение интерфейса не задается и должно быть разработано студентом.

Тесты должны включать заполнение множества некоторым набором значений и проверку работы всех реализованных методов в различных корректных и некорректных ситуациях.

Задание 1.11. *Расширяемое битовое множество.*

Требуется реализовать хранение и работу с подмножествами целых чисел, принадлежащих заданному диапазону. Идея реализации: заводится массив целых чисел, в котором каждому возможному элементу множества сопоставляется один бит. Таким образом, работа сводится к установке или проверке значения отдельного бита в некотором элементе целочисленного массива. При размещении значений, выходящих из заданного диапазона, диапазон множества расширяется автоматически (следует аккуратно продумать алгоритм расширения диапазона, чтобы не сильно снизить эффективность работы).

Реализация класса должна поддерживать следующие возможности:

— создание пустого множества для заданного диапазона чисел;
— очистка всего множества;
— добавление / удаление заданного значения;
— проверка принадлежности заданного значения множеству;
— получение максимального и минимального значений, содержащихся в множестве;
— итератор по множеству (возможность последовательного перебора всех значений из множества);
— объединение, пересечение и другие операции с аналогичным множеством;
— получение границ текущего диапазона множества.

Формальное определение интерфейса не задается и должно быть разработано студентом.

Тесты должны включать заполнение множества некоторым набором значений и проверку работы всех реализованных методов в различных корректных и некорректных ситуациях.

Задание 1.12. *Двумерное битовое множество.*

Требуется реализовать хранение и работу с подмножествами пар (x, y) целых чисел, принадлежащих заданным диапазонам $[x1, x2] \times [y1, y2]$. Идея реализации: сначала паре (x, y) сопоставляется линейный номер (как при записи матрицы по строкам в одномерный массив), затем заводится массив целых чисел, в котором каждому возможному линейному номеру сопоставляется один бит. Таким образом, работа сводится к установке или проверке значения отдельного бита в некотором элементе целочисленного массива. Значения, выходящие из заданного диапазона, не могут размещаться в множестве (сообщение об ошибке), но диапазон можно изменить отдельной функией.

Реализация класса должна поддерживать следующие возможности:

— создание пустого множества для заданного диапазона чисел;
— очистка всего множества;
— добавление / удаление заданного значения;
— проверка принадлежности заданного значения множеству;
— получение max и min значений, содержащихся в множестве (ограничивающий прямоугольник);
— итератор по множеству (возможность последовательного перебора всех значений из множества);
— объединение, пересечение и другие операции с аналогичным множеством;
— изменение диапазона допустимых элементов;
— получение границ текущего диапазона множества.

Формальное определение интерфейса не задается и должно быть разработано студентом.

Тесты должны включать заполнение множества некоторым набором значений и проверку работы всех реализованных методов в различных корректных и некорректных ситуациях.

Задание 1.13. *Полигональная область.*

Требуется реализовать хранение и работу с областью точек $\mathbf{R}^2$, ограниченную заданной ломаной линией. Область не обязательно выпуклая и односвязная, может состоять из нескольких связных компонент.

Идея реализации: хранится список связных компонент области, а каждая связная компонента представлена списком точек вершин ее границы.

Реализация класса должна поддерживать следующие возможности:
— создание пустой области;
— создание односвязной области по заданным массивам координат вершин;
— объединение, пересечение с другой областью;
— изменение координат отдельно взятой точки вершины;
— добавление новой точки на ребро;
— удаление вершины (замена двух смежных ребер на одно);
— положение данной точки относительно области (внутри, вне, на границе);
— итератор по связным компонентам и их вершинам;
— получить количество связных компонент и количество вершин в них.

Формальное определение интерфейса не задается и должно быть разработано студентом.

Тесты должны включать создание некоторых областей и проверку работы всех реализованных методов в различных корректных и некорректных ситуациях.

Задание 1.14. *Выпуклая оболочка.*

Требуется реализовать хранение и работу с выпуклой оболочкой множества точек $\mathbf{R}^2$.

Идея реализации: хранится два списка — список точек, образующих ломаную линию выпуклой оболочки, и список всех остальных точек множества.

Реализация класса должна поддерживать следующие возможности:
— создание пустого множества;
— создание выпуклой оболочки по заданному множеству (массиву) точек;
— добавление / удаление точки из множества;
— получить количество точек в выпуклой оболочке;
— получить массив точек выпуклой оболочки;
— итератор по точкам выпуклой оболочки;
— итератор по точкам множества, не принадлежащим границе выпуклой оболочки;
— расположение данной точки относительно выпуклой оболочки (внутри, вне, на границе);
— получить характеристики выпуклой оболочки (площадь, периметр, min и max координаты и т.п.)

Формальное определение интерфейса не задается и должно быть разработано студентом.

Тесты должны включать создание некоторых оболочек и проверку работы всех реализованных методов в различных корректных и некорректных ситуациях.

Задание 1.15. *Контейнер строк.*

Требуется построить контейнер для размещения текстовых строк без операции удаления. Для хранения выделяется блок памяти, в котором последовательно размещаются поступающие строки. Если очередная строка не помещается в остатке блока, то выделяется следующий блок. Все блоки завязываются в список для осуществимости операции очистки контейнера.

Реализация класса должна поддерживать следующие возможности:
— создание пустого контейнера;
— очистка всего контейнера;
— добавление строки в контейнер (на выходе — указатель на место размещения строки);
— получение количества строк, хранящихся в контейнере;

Формальное определение интерфейса не задается и должно быть разработано студентом.

Тесты должны включать заполнение контейнера некоторым набором строк и проверку работы всех реализованных методов в различных корректных и некорректных ситуациях.

Задание 1.16. *Строковый буфер.*

Требуется реализовать возможность работы со строками произвольной длины. Строка хранится отдельными кусками фиксированной длины, которые связаны в список и добавляются или удаляются по мере необходимости.

Реализация класса должна поддерживать следующие возможности:
— создать пустой буфер или буфер, инициализированный данной строкой;
— очистить строковый буфер;
— добавить заданную строку в конец буфера;
— вставить заданную строку с заданной позиции буфера;
— получить длину строки, накопленной в буфере;
— скопировать строку буфера в заданный символьный массив (по заданному указателю);
— скопировать подстроку из буфера в заданный символьный массив (по заданному указателю);
— получить / изменить символ в заданной позиции;
— заменить одну подстроку на другую в буфере;
— искать заданный символ / подстроку в буфере;
— обрезать накопленную строку по указанной длине;

Формальное определение интерфейса не задается и должно быть разработано студентом.

Тесты должны включать заполнение буфера некоторыми строками и проверку работы всех реализованных методов в различных корректных и некорректных ситуациях.

Задание 1.17. *Буфер чтения строк.*

Требуется реализовать возможность чтения строк произвольной длины из текстового файла. Строка считывается отдельными блоками ограниченной длины, которые связываются в список.

Реализация класса должна поддерживать следующие возможности:
— начать работу с заданным файлом (по имени или указателю FILE*);
— прочитать очередную строку;
— получить длину прочитанной строки;
— скопировать прочитанную строку всю или частично по заданному указателю char*;
— искать заданный символ / подстроку в прочитанной строке;

Формальное определение интерфейса не задается и должно быть разработано студентом.

Тесты должны включать чтение разнообразных строк и проверку работы всех реализованных методов в различных корректных и некорректных ситуациях.

Задание 1.18. *Токенайзер текстовой строки.*

Требуется разбить данную строку на токены по заданному набору символов-разделителей и обеспечить работу с этими токенами. Наряду с исходной строкой, в реализации создается список, хранящий позиции и длины выделенных токенов.

Реализация класса должна поддерживать следующие возможности:
— инициализировать токенайзер данным набором символов-разделителей;
— разбить данную строку на токены;
— получить общее количество найденных токенов;
— возможность последовательного просмотра токенов (итератор);
— получение токена с заданным порядковым номером;
— возможность изменить набор символов-разделителей для данной строки.

Формальное определение интерфейса не задается и должно быть разработано студентом.

Тесты должны включать разбиение разнообразных строк по различным разделителям и проверку работы всех реализованных методов в различных корректных и некорректных ситуациях.

Задание 1.19. *Загрузка config-файла.*

Требуется построить парсер config-файла с операциями чтения значения по имени параметра и возможностью комментариев. Комментарии задаются от символа # до конца строки. Содержательные строки имеют вид

```
имя = значение
имя + значение
```

Имя параметра не содержит пробелов и знака =, значение — это оставшаяся часть строки после знака равенства, исключая пробельные символы в начале и конце. В первом случае значение сопоставляется указанному имени, а во втором — добавляется в конец ранее сопоставленного значения.

Реализация класса должна поддерживать следующие возможности:
— загрузка параметров из указанного config-файла;
— получение строк с диагностикой возможных ошибок при разборе входного файла;
— получение строки-значения, соответствующей имени параметра;
— объединение с множеством параметров, хранящихся в другом аналогичном классе.

Формальное определение интерфейса не задается и должно быть разработано студентом.

Тесты должны включать заполнение множества параметров из некоторых файлов и проверку работы всех реализованных методов в различных корректных и некорректных ситуациях.

Задание 1.20. *Загрузка config-файла с предобработкой.*

Пусть имеется конфигурационный файл, в котором комментарии задаются от символа # до конца строки, а содержательные строки имеют вид

```
имя = значение
```

Имя параметра не содержит пробелов и знака =, значение — это оставшаяся часть строки после знака равенства, исключая пробельные символы в начале и конце. При этом, если в значении последующих строк встретится выражение `$имя$`, то его следует заменить на `значение`. Например,

```
par1 = abcdef            заменяется на        par1 = abcdef
par2 = 123$par1$456                           par2 = 123abcdef456
```

Реализация класса должна поддерживать следующие возможности:
— загрузка параметров из указанного config-файла;
— получение строк с диагностикой возможных ошибок при разборе входного файла;
— получение развернутой строки-значения, соответствующей имени параметра;
— объединение с множеством параметров, хранящихся в другом аналогичном классе.

Формальное определение интерфейса не задается и должно быть разработано студентом.

Тесты должны включать заполнение множества параметров из некоторых файлов и проверку работы всех реализованных методов в различных корректных и некорректных ситуациях.

Задание 1.21. *Разреженная матрица фиксированного размера.*

Требуется реализовать числовую $m \times n$ матрицу, в которой количество ненулевых элементов значительно меньше mn. Для этого каждая строка матрицы представляется списком, хранящим значения элементов вместе с их j-индексами. Такое списки собраны в массив, т.е. i-й элемент этого массива представляет собой список i-й строки матрицы. В данной реализации хранятся только ненулевые элементы матрицы, т.е., если элемент с индексами i, j отсутствует, он считается равным нулю.

Реализация класса должна обеспечивать следующие возможности:
— создать матрицу на заданные размерности;
— установить / получить значение элемента с заданными индексами;
— обменять местами две указанные строки матрицы;
— умножить строку матрицы на данное число;
— выполнить линейную комбинацию данной строки с другой строкой;
— получить вектор столбец / строку матрицы для данного индекса;

Формальное определение интерфейса не задается и должно быть разработано студентом.

Тесты должны включать заполнение матрицы некоторыми значениями и проверку работы всех реализованных методов в различных корректных и некорректных ситуациях. Одним из тестов может быть решение системы линейных уравнений методом Гаусса.

Задание 1.22. *Разреженная матрица переменного размера.*

Требуется реализовать числовую матрицу, в которой количество ненулевых элементов значительно меньше общего количества элементов. Для этого каждая строка матрицы представляется списком, хранящим значения элементов вместе с их j-индексами. Такое списки сами собраны в список, упорядоченный по номеру строки, т.е. каждый элемент этого списка хранит указатель на список-строку и соответствующий ей i-индекс. В данной реализации хранятся только ненулевые элементы матрицы, т.е., если элемент с индексами i, j отсутствует, он считается равным нулю. Таким образом, размеры матрицы определяются по значениям максимальных индексов i, j.

Реализация класса должна обеспечивать следующие возможности:
— создать (пустую) матрицу;
— получить текущие размеры матрицы;
— установить / получить значение элемента с заданными индексами;
— обменять местами две указанные строки матрицы;
— выполнить линейную комбинацию данной строки с другой строкой;
— вычислить сумму элементов i-й строки для $j_1 \le j \le j_2$;
— получить подматрицу по заданному диапазону или множеству индексов строк и столбцов.

Формальное определение интерфейса не задается и должно быть разработано студентом.

Тесты должны включать заполнение матрицы некоторыми значениями и проверку работы всех реализованных методов в различных корректных и некорректных ситуациях. Одним из тестов может быть моделирование простейшей электронной таблицы, содержащей столбец с суммой столбцов и строку с суммой строк.

Задание 1.23. *Стек строк с файловым буфером I.*

Требуется реализовать работу со строками произвольной длины по стековому принципу с ограниченным использованием оперативной памяти. Идея реализации: выделяется блок памяти и строки укладываются в него одна за другой. Если для добавления новой строки не хватает места, то заполненный блок записывается в файл, а освободившееся место используется для новых добавлений. Если при извлечении строк буфер памяти окажется пустым, то он заполняется чтением ранее сохраненной в файле информации. Таким образом, в реализации есть два стека — стек строк в памяти и стек блоков записей в файле.

Реализация класса должна поддерживать следующие возможности:
— создать пустой стек;
— добавить строку;
— получить длину строки на вершине стека;
— копировать строку на вершине стека по заданному указателю;
— удалить вершину стека;
— получить общее количество строк в стеке;
— очистить стек;
— сохранить состояние стека в файле / загрузить из файла.

Формальное определение интерфейса не задается и должно быть разработано студентом.

Тесты должны включать заполнение стека некоторыми строками и проверку работы всех реализованных методов в различных корректных и некорректных ситуациях.

Задание 1.24. *Стек строк с файловым буфером II.*

Требуется реализовать работу со строками произвольной длины по стековому принципу с ограниченным использованием оперативной памяти. Идея реализации: поступающие строки укладываются в список. Когда суммарная длина накопленных строк превысит заданное значение, все строки, кроме текущей вершины, переписываются единой записью в файл. Если после извлечения текущей верщины в памяти не окажется строк, то из файла считывается ранее запомненная запись и из нее формируется исходный список строк Таким образом, в реализации существует как бы два стека — один стек строк в памяти, реализованный на базе списка и второй — стек записей в файле.

Реализация класса должна поддерживать следующие возможности:
— создать пустой стек;
— добавить строку;
— получить длину строки на вершине стека;
— копировать строку на вершине стека по заданному указателю;
— удалить вершину стека;
— получить общее количество строк в стеке;
— очистить стек;
— сохранить состояние стека в файле / загрузить из файла.

Формальное определение интерфейса не задается и должно быть разработано студентом.

Тесты должны включать заполнение стека некоторыми строками и проверку работы всех реализованных методов в различных корректных и некорректных ситуациях.

Задание 1.25. *Очередь строк с отметкой времени.*

Требуется реализовать работу очереди строк, причем каждый элемент очереди дополнительно хранит метку времени добавления этого элемента в очередь. Очередь реализуется на базе списка и тем самым не имеет ограничений на максимальное количество элементов. Строки размещаются в памяти независимо, а указатели на них сохраняютя в очереди.

Реализация класса должна поддерживать следующие возможности:
— создать пустую очередь;
— добавить строку (в конец);
— получить длину строки в начале очереди;
— копировать строку в начале очереди по заданному указателю;
— удалить начало очереди;
— получить общее количество строк в очереди;
— очистить очередь;
— получить метку времени в начале очереди.

Формальное определение интерфейса не задается и должно быть разработано студентом.

Тесты должны включать заполнение очереди некоторыми строками и проверку работы всех реализованных методов в различных корректных и некорректных ситуациях.

Задание 1.26. *Кольцевая очередь строк.*

Требуется реализовать работу очереди строк на базе одного отрезка памяти, причем при полном заполнении этого отрезка новые добавляемые строки затирают (целиком) строки из начала очереди. Все строки размещаются в кольцевом буфере памяти без промежутков (при этом одна из строк может оказаться физически разорванной). Указатели на строки очереди можно дополнительно хранить в отдельном списке.

Реализация класса должна поддерживать следующие возможности:
— создать пустую очередь на заданное количество байт;
— добавить строку (в конец);
— получить длину строки в начале очереди;
— копировать строку из начала очереди по заданному указателю;
— удалить начало очереди;
— получить общее количество строк в очереди;
— очистить очередь;
— получить количество потерянных строк в начале очереди

Формальное определение интерфейса не задается и должно быть разработано студентом.

Тесты должны включать заполнение очереди некоторыми строками и проверку работы всех реализованных методов в различных корректных и некорректных ситуациях.

Задание 1.27. *Скользящее приемное окно, как в TCP.*

Пусть мы имеем набор строк, занумерованных по порядку целыми числами. Эти строки вместе со своими номерами передаются в случайном порядке на хранение в класс, похожий на очередь и пытающийся разместить строки по порядку их номеров. Из этой очереди можно забирать только группы строк с постедовательными номерами. Например, если на хранение поступили строки с номерами 5, 0, 2, 4, 1, то мы можем забрать строки 0, 1, 2, а строки 4 и 5 взять не можем так как перед ними пока отсутствуют строка с номером 3. Как только она появится, можно будет забрать строки 3, 4, 5.

Реализация класса должна поддерживать следующие возможности:
— создать пустую очередь;
— добавить на хранение строку с указанным номером;
— получить суммарную длину и количество строк в связном фрагменте в начале очереди;
— взять (копировать и удалить) строку из начала очереди;
— получить порядковый номер строки, расположенной в начале очереди;
— получить максимальный номер добавленной строки.

Формальное определение интерфейса не задается и должно быть разработано студентом.

Тесты должны включать заполнение очереди некоторыми строками и проверку работы всех реализованных методов в различных корректных и некорректных ситуациях.