Требования к программам

1. Программа работает с однонаправленным списком объектов типа `student`:

```
class student
  {
  private:
    char * name;
    int    value;
  public:
    ...
  };
class list_node : public student
  {
  private:
    list_node * next;
  public:
    ...
    friend class list;
  };
class list
  {
  private:
    list_node * head;
  public:
    ...
    int read (const char * filename);
    void print (int r);
    int get_less_than_previous ();
  };
```

   Все функции в задании являются членами класса "список".

2. Программа должна получать все параметры в качестве аргументов командной строки. Аргументы командной строки:

   1) *r* – количество выводимых значений в списке,
   2) `filename` – имя файла, откуда надо прочитать список.

   Например, запуск

```
./a.out 4 a.txt
```

   означает, что список прочитать из файла `a.txt`, выводить не более 4-х элементов списка.

3. Класс "список" должен содержать функцию ввода списка из указанного файла.

4. Ввод списка из файла. В указанном файле находится список в формате:

Слово-1	Целое-число-1
Слово-2	Целое-число-2
…	…
Слово-n	Целое-число-n

где слово – последовательность алфавитно-цифровых символов без пробелов. Концом ввода считается конец файла. Программа должна выводить сообщение об ошибке, если указанный файл не может быть прочитан или содержит данные неверного формата.

5. Решение задачи должно быть оформлено в виде подпрограммы, находящейся в отдельном файле. Получать в этой подпрограмме дополнительную информацию извне через глобальные переменные, включаемые файлы и т.п. запрещается.

6. Класс "список" должен содержать подпрограмму вывода на экран списка длины не более r. Эта подпрограмма используется для вывода исходного списка после его инициализации, а также для вывода на экран результата. Подпрограмма выводит на экран не более, чем r элементов списка, где r – параметр этой подпрограммы (аргумент командной строки). Каждый элемент списка должен печататься на новой строке.

7. Класс "список" должен содержать функцию количества элементов, меньших предыдущего элемента. Результат этой функции выводится на экран для результирующего списка.

8. Программа должна выводить на экран время, затраченное на решение.

Задачи

1. Написать подпрограмму – член класса "список", сортирующую список методом *"пузырька"*:

   Последовательным просмотром элементов списка найти первый элемент, больший следующего. Поменять их местами и продолжить просмотр до конца списка. Тем самым наибольший элемент передвинется на последнее место. Переместить этот элемент в начало нового списка, укоротив исходный на 1. Следующие просмотры начинать опять сначала, идя до конца (уже укороченного списка) и перекладывая затем последний элемент в начало нового списка. Массив будет упорядочен после перекладывания последнего элемента в новый список и сокращения длины исходного до 0. Указатель на этот элемент будет новым началом отсортированного списка.

   Общее количество сравнений в наихудшем случае не должно превышать $n^2/2+O(n)$, а перестановок ссылок между элементами списка – $3n^2/2+O(n)$.

2. Написать подпрограмму – член класса "список", сортирующую список методом *нахождения минимума*:

   Найти элемент списка, имеющий наименьшее значение, переместить его в начало нового списка, уменьшив длину исходного списка на 1. Затем просмотром (уже укороченного списка) найти в нем минимальный элемент и переместить его в конец нового списка, уменьшив длину исходного списка на 1. Так будем повторять до сокращения длины исходного до 0. Начало нового списка будет началом результирующего отсортированного списка.

   Общее количество сравнений в наихудшем случае не должно превышать $n^2/2+O(n)$, а перестановок ссылок между элементами списка – $n+O(1)$.

3. Написать подпрограмму – член класса "список", сортирующую список методом *линейной вставки*:

   Перекладываем первый элемент списка в начало нового списка, уменьшив длину исходного списка на 1. Новый список (длиной 1) отсортирован. Берем первый элемент исходного (уже укороченного) списка и линейным просмотром находим место его вставки в упорядоченный новый список так, чтобы его упорядоченность сохранилась. Вставляем элемент в найденную позицию, удалив его из исходного списка. Так будем повторять до сокращения длины исходного до 0. Начало нового списка будет началом результирующего отсортированного списка.

   Общее количество сравнений наихудшем случае не должно превышать $n^2/2+O(n)$, а перестановок ссылок между элементами списка – $n+O(1)$.

4. Написать подпрограмму – член класса "список", сортирующую список методом слияний (*сортировка Неймана*):

   Вначале весь список рассматривается как совокупность упорядоченных групп по одному элементу в каждом. Отрезаем две группы, укорачивая исходный список на 2 и создавая два новых списка A и B. Переставляя ссылки между элементами этих списков, так, чтобы получился объединенный упорядоченный список, получаем новый список $C = A \cup B$. Продолжаем этот процесс отрезания групп от исходного списка (кроме, может быть, последней группы, которой не нашлось парной). создавая два новых списка A и B, которые объединяются в упорядоченный список, который по окончании слияния добавляется в конец нового списка. Этот процесс завершается, когда длина исходного списка станет равна 0. Затем этот "новый" список называется "исходным" и процесс укрупнения упорядоченных групп по 2 элемента до групп по 4 элемента и т.д. продолжается. Процесс завершается, когда весь список находится в одной упорядоченной группе.

   Общее количество сравнений и перестановок ссылок между элементами списка в наихудшем случае не должно превышать $n\log_2 n + O(n)$.

5. Написать подпрограмму – член класса "список", сортирующую список методом *"быстрой" сортировки*:

   Создаем два пустых списка (нулевые указатели на начало): L и U. За один проход по исходному списку находим указатель на элемент, стоящий приблизительно в середине списка (например, полагаем два указателя M и T равными началу списка, затем, пока T не дойдет до конца списка, продвигаем указатель T на две позиции вперед, а указатель M – на одну). Вырезаем элемент, на который указывает M (т.е. $*M$), из исходного списка. Затем за один проход по исходному списку перемещаем элементы, меньшие $*M$, в список L, элементы большие $*M$, в список U, остальные (т.е. равные $*M$) добавляем в конец списка, началом которого будет элемент M. По окончании этой процедуры все элементы списка L будут меньше элемента $*M$, все элементы списка U будут больше элемента $*M$, все элементы списка списка M будут равны элементу $*M$. Затем сортируем каждый из списков L и U этим же алгоритмом. Для уменьшения глубины рекурсии вначале сортируем более короткий список, а вместо повторного вызова процедуры для более длинного списка переходим к ее началу с новым значениями указателя на начало "исходного" списка. Для упрощения сборки итогового списка можно склеить списки L, M и U в один, а в этой процедуре ввести аргумент, по умолчанию равный 0, указывающий на конец списка.

   Общее количество сравнений в наихудшем случае не должно превышать $n^2/2 + O(n)$, а перестановок ссылок между элементами списка – $3n^2/2 + O(n)$.