ТРЕБОВАНИЯ К ПРОГРАММАМ

1. Программа работает с массивом объектов типа `student`:

```
class student
  {
  private:
    char * name;
    int    value;
  public:
    ...
  };
```

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

   1) n – размерность массива,
   2) r – количество выводимых значений в массиве,
   3) s – задает номер формулы для инициализации массива, должен быть равен 0 при вводе массива из файла
   4) `filename` – имя файла, откуда надо прочитать массив. Этот аргумент **отсутствует**, если $s! = 0$.

   Например, запуск

```
./a.out 4 4 0 a.txt
```

   означает, что массив длины 4 надо прочитать из файла `a.txt`, и выводить не более 4-х элементов масива, а запуск

```
./a.out 1000000 6 1
```

   означает, что массив длины 1000000 надо инициализировать по формуле номер 1, и выводить не более 6-ти элементов массива.

3. Ввод массива должен быть оформлен в виде подпрограммы, находящейся в отдельном файле.

4. Ввод массива из файла. В указанном файле находится массив в формате:

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

   где n - указанный размер массива, слово – последовательность алфавитно-цифровых символов без пробелов. Программа должна выводить сообщение об ошибке, если указанный файл не может быть прочитан, содержит меньшее количество данных или данные неверного формата.

5. Ввод массива по формуле. Элемент a_i массива A полагается равным

$$a_i = \{\text{"Student"}, f(s,n,i)\}, \quad i = 1,\ldots,n,$$

где $f(s,n,i)$ - функция, которая возвращает значение (i)-го элемента массива по формуле номер s (аргумент командной строки). Функция $f(s,n,i)$ должна быть оформлена в виде отдельной подпрограммы.

$$f(s,n,i) = \begin{cases} i & \text{при} \quad s=1 \\ n-i & \text{при} \quad s=2 \\ i/2 & \text{при} \quad s=3 \\ n-i/2 & \text{при} \quad s=4 \end{cases}$$

6. Решение задачи должно быть оформлено в виде подпрограммы, находящейся в отдельном файле и получающей в качестве аргументов массив и его длину (в 1 – 3 задачах также дополнительные аргументы). Получать в этой подпрограмме дополнительную информацию извне через глобальные переменные, включаемые файлы и т.п. запрещается.

7. Программа должна содержать подпрограмму вывода на экран массива длины не более r. Эта подпрограмма используется для вывода исходного массива после ее инициализации, а также для вывода на экран результата. Подпрограмма выводит на экран не более, чем r элементов массива, где r – параметр этой подпрограммы (аргумент командной строки). Каждый элемент массива должен печататься на новой строке.

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

Задачи

1. Написать функцию, получающую в качестве аргументов неубывающий массив $a[n]$ объектов типа `student`, целое число n, являющееся длиной этого массива и объект типа `student` x, и возвращающую место, где в этот массив можно вставить x (т.е. целое число i такое, что $a[i] \le x \le a[i+1]$). Место определяется двоичным поиском по следующему алгоритму (*алгоритм деления пополам*).

   Взять первоначально 0 и n в качестве границ поиска элемента; далее, до тех пор, пока границы не совпадут, шаг за шагом сдвигать эти границы следующим образом: сравнить x с $a[s]$, где s – целая часть среднего арифметического границ; если $a[s] < x$, то заменить прежнюю нижнюю границу на $s+1$, а верхнюю оставить без изменений, иначе оставить без изменения нижнюю границу, а верхнюю заменить на s; когда границы совпадут, став равными некоторому числу t, выполнение закончится с результатом t.

   Общее количество сравнений и перестановок элементов в наихудшем случае не должно превышать $\log_2 n + O(1)$. Основная программа должна заполнять данными массив, выводить его на экран, вводить с командной строки x, вызывать эту функцию и выводить на экран результат ее работы.

2. Написать подпрограмму, получающую в качестве аргументов три массива $a[n]$, $b[m]$, $c[n+m]$ объектов типа `student`, где $a[n]$, $b[m]$ не убывают, и целые числа n и m, и строящую по неубывающим массивам $a[n]$, $b[m]$ неубывающий массив $c[n+m]$ слиянием первых двух за $n+m+O(1)$ сравнений и $n+m+O(1)$ пересылок элементов по следующему алгоритму

   Просматриваем очередные элементы $a[i]$, $i=0,\ldots,n-1$ и $b[j]$, $j=0,\ldots,m-1$ массивов a и b. Если $a[i] < b[j]$, то $c[k] = a[i]$ и увеличиваем i на 1, иначе $c[k] = b[j]$ и увеличиваем j на 1. Здесь k, $k=0,\ldots,n+m-1$ – очередной элемент массива c (здесь всегда равен $i+j$).

   Основная программа должна заполнять данными массивы $a[n]$ и $b[m]$, выводить их на экран, вызывать эту подпрограмму и выводить на экран результат ее работы – массив $c[n+m]$.

3. Написать функцию, получающую в качестве аргументов неубывающий массив $a[n]$ объектов типа `student`, целое число n, являющееся длиной этого массива и объект типа `student` x, и переставляющую элементы массива так, чтобы вначале шли все элементы, меньшие x, а затем элементы, большие x. Функция возвращает место, где в этот массив можно вставить x (т.е. целое число i

такое, что $a[i-1] \le x \le a[i]$, если x меньше всех элементов массива a, то $i = 0$, если больше — то $i = n$). Решение задачи производится следующим алгоритмом:

Просматривая массив с начала (линейным поиском), находим i такое, что $a[i] \ge x$. Просматривая массив с конца (линейным поиском), находим j такое, что $a[j] < x$. Если $i \le j$, то меняем $a[i]$ и $a[j]$ местами, i увеличиваем на 1, j уменьшаем на 1. Повторяем эту процедуру пока $i \le j$. После этого все элементы $a[0], \ldots, a[i]$ будут не больше всех элементов $a[j], \ldots, a[n-1]$. Возвращаемым значением функции будет $i = j$.

Общее количество сравнений элементов в наихудшем случае не должно превышать $n/2 + O(1)$, а пересылок элементов – $3n/2 + O(1)$. Основная программа должна заполнять данными массив, выводить его на экран, вводить с командной строки x, вызывать эту функцию и выводить на экран результат ее работы: массив $a[n]$ и значение $i = j$.

4. Написать подпрограмму, получающую в качестве аргументов массив $a[n]$ объектов типа `student` и целое число n, и переставляющую элементы массива $a[n]$ в неубывающем порядке: $a[0] \le a[1] \le \cdots \le a[n-1]$ по следующему алгоритму (*"пузырьковая"сортировка*):

   Последовательным просмотром чисел $a[0], a[1], \ldots, a[n-1]$ найти наименьшее i такое, что $a[i] > a[i+1]$. Поменять $a[i]$ и $a[i+1]$ местами, возобновить просмотр с элемента $a[i+1]$ и т.д. Тем самым наибольший элемент передвинется на последнее место. Следующие просмотры начинать опять сначала, уменьшая на 1 количество просматриваемых элементов. Массив будет упорядочен после просмотра, в котором участвовали только первый и второй элементы.

   Общее количество сравнений в наихудшем случае не должно превышать $n^2/2 + O(n)$, а пересылок элементов – $3n^2/2 + O(n)$. Основная программа должна заполнять данными массив, выводить его на экран, вызывать эту подпрограмму и выводить на экран результат ее работы.

5. Написать подпрограмму, получающую в качестве аргументов массив $a[n]$ объектов типа `student` и целое число n, и переставляющую элементы массива $a[n]$ в неубывающем порядке: $a[0] \le a[1] \le \cdots \le a[n-1]$ по следующему алгоритму (*сортировка нахождением минимума*):

   Найти элемент массива, имеющий наименьшее значение, переставить его с первым элементом, затем проделать то же самое, начав со второго элемента и т.д.

   Общее количество сравнений в наихудшем случае не должно превышать $n^2/2 + O(n)$, а пересылок элементов – $3n + O(1)$. Основная программа должна заполнять данными массив, выводить его на экран, вызывать эту подпрограмму и выводить на экран результат ее работы.

6. Написать подпрограмму, получающую в качестве аргументов массив $a[n]$ объектов типа `student` и целое число n, и переставляющую элементы массива $a[n]$ в неубывающем порядке: $a[0] \le a[1] \le \cdots \le a[n-1]$ по следующему алгоритму (*сортировка линейной вставкой*):

   Просматривать последовательно $a[1], \ldots, a[n-1]$ и каждый новый элемент $a[i]$ вставлять на подходящее место в уже упорядоченную совокупность $a[0], \ldots, a[i-1]$. Это место определяется последовательным сравнением $a[i]$ с упорядоченными элементами $a[0], \ldots, a[i-1]$.

   Общее количество сравнений и пересылок элементов в наихудшем случае не должно превышать $n^2/2 + O(n)$ (каждое). Основная программа должна заполнять данными массив, выводить его на экран, вызывать эту подпрограмму и выводить на экран результат ее работы.

7. Написать подпрограмму, получающую в качестве аргументов массив $a[n]$ объектов типа `student` и целое число n, и переставляющую элементы массива $a[n]$ в неубывающем порядке: $a[0] \le a[1] \le \cdots \le a[n-1]$ по следующему алгоритму (*сортировка двоичной вставкой*):

   Просматривать последовательно $a[1], \ldots, a[n-1]$ и каждый новый элемент $a[i]$ вставлять на подходящее место в уже упорядоченную совокупность $a[0], \ldots, a[i-1]$. Это место определяется алгоритмом деления пополам (см. задачу 1).

Общее количество сравнений в наихудшем случае не должно превышать $n\log_2 n + O(n)$, а пересылок элементов – $n^2/2 + O(n)$. Основная программа должна заполнять данными массив, выводить его на экран, вызывать эту подпрограмму и выводить на экран результат ее работы.

8. Написать подпрограмму, получающую в качестве аргументов массив $a[n]$ и вспомогательный массив $b[n]$ объектов типа `student`, и целое число n, и переставляющую элементы массива $a[n]$ в неубывающем порядке: $a[0] \le a[1] \le \cdots \le a[n-1]$ по следующему алгоритму (*сортировка Неймана*):

   Вначале весь массив рассматривается как совокупность упорядоченных групп по одному элементу в каждом. Слиянием соседних групп (см. задачу 2) получаются упорядоченные группы, каждая из которых содержит два элемента (кроме, может быть, последней группы, которой не нашлось парной). Далее упорядоченные группы укрупняются тем же способом и т.д. Используется вспомогательный массив b[n].

   Общее количество сравнений и пересылок элементов в наихудшем случае не должно превышать $n\log_2 n + O(n)$ (каждое). Основная программа должна заполнять данными массив, выводить его на экран, вызывать эту подпрограмму и выводить на экран результат ее работы.

9. Написать подпрограмму, получающую в качестве аргументов массив $a[n]$ объектов типа `student` и целое число n, и переставляющую элементы массива $a[n]$ в неубывающем порядке: $a[0] \le a[1] \le \cdots \le a[n-1]$ по следующему алгоритму (*"быстрая"сортировка*):

   Полагаем $x = a[n/2]$. Алгоритмом задачи 3 находим место, где в этот массив можно вставить x, т.е. такое i, что все элементы $a[0],\ldots,a[i-1]$ будут меньше или равны всех элементов $a[i],\ldots,a[n-1]$. Затем сортируем каждую из частей (т.е. $a[0],\ldots,a[i-1]$ и $a[i],\ldots,a[n-1]$) этим же алгоритмом. Для уменьшения глубины рекурсии вначале сортируем более короткий массив. Затем вместо повторного вызова процедуры переходим к ее началу с новыми значениями указателя на начало массива a и его длины n.

   Общее количество сравнений в наихудшем случае не должно превышать $n^2/2 + O(n)$, а пересылок элементов – $3n^2/2 + O(n)$. Основная программа должна заполнять данными массив, выводить его на экран, вызывать эту подпрограмму и выводить на экран результат ее работы.

10. Написать подпрограмму, получающую в качестве аргументов массив $a[n]$ объектов типа `student` и целое число n, и переставляющую элементы массива $a[n]$ в неубывающем порядке: $a[0] \le a[1] \le \cdots \le a[n-1]$ по следующему алгоритму (*"турнирная"сортировка или алгоритм heapsort*):

    Массив $a[n]$ рассматриваем как бинарное дерево с корнем $a[0]$. Для элемента с номером $a[k]$ потомками являются элементы $a[2*k+1]$ и $a[2*k+2]$, а родителем – элемент $a[(k-1)/2]$. На первом этапе алгоритма превращаем наше бинарное дерево в т.н. упорядоченную пирамиду, т.е. в дерево, в котором всякая цепочка от корня до любого конечного элемента выстроена по убыванию. Для этого для всех $k = 1,\ldots,n-1$ идем от элемента $a[k]$ по цепочке его родителей, продвигая его на свое место (подобно тому, как это делалось в алгоритме сортировки линейной вставкой). После этого этапа алгоритма в корне дерева (т.е. в $a[0]$) будет находиться максимальный элемент массива. На втором этапе алгоритма для всех $k = n-1,\ldots,1$: меняем корень (т.е. $a[0]$) и элемент $a[k]$ местами и рассматриваем массив a как имеющий длину k. В этом массиве восстанавливаем структуру упорядоченной пирамиды, нарушенную помещением элемента $a[k]$ в $a[0]$. Для этого идем от элемента $a[0]$ по цепочкам его потомков, продвигая его на свое место (подобно тому, как это делалось в алгоритме сортировки линейной вставкой, с одним отличием: элемент пирамиды должен быть больше обоих своих потомков).

    Общее количество сравнений в наихудшем случае не должно превышать $4n\log_2 n + O(n)$, а пересылок элементов – $2n\log_2 n + O(n)$. Основная программа должна заполнять данными массив, выводить его на экран, вызывать эту подпрограмму и выводить на экран результат ее работы.