Требования к программам

1. В программе должны быть реализованы следующие структуры данных:

   - Контейнер данных объектов типа `student`:

```
enum class io_status
{
  success,
  eof,
  format,
  memory,
};

class student
  {
  private:
    char * name = nullptr;
    int    value = 0;
  public:
    student () = default;
    ...
    void print (FILE *fp = stdout) const;
    io_status read (FILE * fp = stdin);
    int get_length () const { return (name != nullptr ? 1 : 0);}
    int operator< (const student& b) const;
  };
```

   - Двунаправленный список объектов типа `student`:

```
class list2;
class list2_node : public student
  {
  private:
    list2_node * next = nullptr;
    list2_node * prev = nullptr;
  public:
    list2_node () = default;
    ...
    friend class list2;
  };
class list2
  {
  private:
    list2_node * head = nullptr;
    static int m;
    static int r;
  public:
    list2 () = default;
    ...
    static void set_m (int m) { list2::m = m; }
    static void set_r (int r) { list2::r = r; }
    void print (FILE *fp = stdout) const;
    io_status read (FILE * fp = stdin);
    int get_length () const; // number of elements 'student'
    int operator< (const list2& b) const;
  };
```

- Операция сравнения `list2::operator<` задается правилом:
  - пустой список меньше любого не пустого,
  - один не пустой список меньше другого не пустого списка, если его первый элемент меньше первого элемента другого списка относительно операции сравнения `student::operator<`
- Функция `list2::read` считывает не более m элементов списка, если `list2` $:: m > 0$
- Функция `list2::print` выводит не более r элементов списка, если `list2` $:: r > 0$

- Бинарное дерево

```
template <class T> class tree;
template <class T>
class tree_node : public T
  {
  private:
    tree_node * left  = nullptr;
    tree_node * right = nullptr;
  public:
    tree_node () = default;
    ...
    friend class tree<T>;
  };
template <class T>
class tree
  {
  private:
    tree_node<T> * root = nullptr;
  public:
    tree () = default;
    ...
    io_status read (FILE * fp = stdin, unsigned int max_read = -1);
    void print (unsigned int r = 10, FILE *fp = stdout) const;
  };
```

2. Все функции в задании являются членами класса "дерево".

3. Все функции в задании вызываются дважды:

   - для бинарного дерева объектов типа `student`, т.е. `tree<student>`
   - для бинарного дерева объектов типа `list2`, т.е. `tree<list2>`

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

   1) r – максимальное количество выводимых уровней в дереве (оно же максимальное число выводимых элементов в списке),
   2) `filename` – имя файла, откуда надо прочитать дерево,
   3) m – количество элементов в списке для заполнения `tree<list2>`.

   Например, запуск

   ```
   ./a.out 4 a.txt 100
   ```

   означает, что дерево надо прочитать из файла `a.txt`, выводить не более 4-х уровней дерева, использовать $m = 100$ для заполнения списка (см. ниже).

5. Класс "дерево" должен содержать функцию ввода дерева из указанного файла.

6. Ввод дерева из файла. В указанном файле находится дерево 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. Длина слова неизвестна, память под него выделяется динамически. При заполнении первый объект типа – элемент дерева попадает в корень дерева, каждый новый объект типа – элемент дерева добавляется в левое поддерево, если он меньше текущего узла относительно операции сравнения в элементе дерева, и в правое поддерево иначе. **Никакие другие функции, кроме функции ввода дерева, не используют упорядоченность дерева**. Концом ввода считается конец файла. Программа должна выводить сообщение об ошибке, если указанный файл не может быть прочитан или содержит данные неверного формата.

7. Класс "дерево" должен содержать подпрограмму вывода на экран не более чем r уровней дерева. Эта подпрограмма используется для вывода исходного дерева после его инициализации. Подпрограмма выводит на экран не более, чем r уровней дерева, где r – параметр этой подпрограммы (аргумент командной строки). Каждый элемент дерева должен печататься на новой строке и так, чтобы структура дерева была понятна.

8. Поскольку дерево не изменяется функциями из задач, кроме последней, то для всех задач надо сделать **одну функцию** `main`, в которой

   - вначале создается объект `tree<student>`, вводится из указанного файла, выводится указанное число уровней на экран, вызываются все функции задач, выводятся результаты и время их работы; вызов функции, реализующей последнюю задачу, должен быть последним и сразу после него надо вывести указанное число уровней преобразованного дерева на экран; затем этот объект удаляется;
   - потом создается объект `tree<list2>`, вводится из указанного файла, выводится указанное число уровней на экран, вызываются все функции задач, выводятся результаты и время их работы; вызов функции, реализующей последнюю задачу, должен быть последним и сразу после него надо вывести указанное число уровней преобразованного дерева на экран; затем этот объект удаляется.

   После компиляции должен получиться один исполняемый файл `a.out`, а не несколько.

9. Схема функции `main`:

```
// Статические члены класса list2
int list2::m = 0;
int list2::r = 0;

int main (int argc, char *argv[])
{
  // Ввод аргументов командной строки
  ...
  // Задание значений статическим членам класса
  list2::set_m (m);
  list2::set_r (r);
  ...
```

```
  tree<student> *a = new tree<student>;
  // Работа с деревом tree<student>
  ...
  delete a;
  tree<list2> *b = new tree<list2>;
  // Работа с деревом tree<list2>
  ...
  delete b;
  return 0;
}
```

10. Вывод результата работы функции в функции `main` должен производиться по формату:

```
printf ("%s : Task = %d M = %d Result = %d Elapsed = %.2f\n",
        argv[0], m, task, res, t);
```

где

- `argv[0]` – первый аргумент командной строки (имя образа программы),
- `m` – параметр *m* командной строки,
- `task` – номер задачи (1–6),
- `res` – результат работы функции, реализующей решение этой задачи,
- `t` – время работы функции, реализующей решение этой задачи.

Вывод должен производиться в точности в таком формате, чтобы можно было автоматизировать обработку запуска многих тестов.

Задачи

1. Написать подпрограмму – член класса "дерево", возвращающую целое значение, равное количеству элементов типа `student` в концевых элементах этого дерева.

2. Написать подпрограмму – член класса "дерево", возвращающую целое значение, равное максимальному числу элементов типа `student` в одной ветви этого дерева.

3. Написать подпрограмму – член класса "дерево", возвращающую целое значение, равное максимальному количеству элементов типа `student` в одном уровне этого дерева.

4. Написать подпрограмму – член класса "дерево", возвращающую целое значение, равное максимальной по модулю разности между количеством элементов типа `student` в левом и правом поддереве в узлах дерева.

5. Написать подпрограмму – член класса "дерево", возвращающую целое значение, равное количеству элементов типа `student` в узлах дерева, имеющих ровно 1 потомка.

6. Написать подпрограмму – член класса "дерево", которая удаляет все узлы дерева, содержащие элемент типа `student` с наименьшим значением поля `value` среди всех узлов дерева. Узел дерева удаляется вместе со всеми потомками (т.е. удаляется поддерево с вершиной в этом узле). Функция возвращает целое значение, равное количеству удаленных узлов в дереве (только самих узлов, содержащих элемент типа `student` с наименьшим значением поля `value`, без учета потомков).