Требования к программам

1. Программа работает с бинарным деревом объектов типа `student`:

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

class student
{
  private:
    char * name  = nullptr;
    int    value = 0;
  public:
    student () = default;
    student (const student& x) = delete;
    student (student&& x)
      {
        name = x.name; x.name = nullptr;
        value = x.value; x.value = 0;
      }
    ~student ()
      {
        erase ();
      }
    student& operator= (const student& x) = delete;
    student& operator= (student&& x);
      {
        if (this == &x)
          return *this;
        erase ();
        name = x.name; x.name = nullptr;
        value = x.value; x.value = 0;
        return *this;
      }
    void print (FILE *fp = stdout) const;
    io_status read (FILE * fp = stdin);
    int operator> (const student& x) const
      { return (cmp (x) > 0 ? 1 : 0); }
    int operator< (const student& x) const
      { return (cmp (x) < 0 ? 1 : 0); }
    int operator== (const student& x) const
      { return (cmp (x) == 0 ? 1 : 0); }

  private:
    io_status init (const char * n, int v);
    void erase ();
    int cmp (const student& x) const;
```

```
}

class tree;
class tree_node : public student
  {
  private:
    tree_node * left  = nullptr;
    tree_node * right = nullptr;
  public:
    tree_node () = default;
    tree_node (const tree_node& x) = delete;
    tree_node (tree_node&& x) : student ((student&&)x)
      {
        erase_links ();
        x.erase_links ();
      }
    ~tree_node ()
      {
        erase_links ();
      }
    tree_node& operator= (const tree_node& x) = delete;
    tree_node& operator= (tree_node&& x)
      {
        if (this == &x)
          return *this;
        (student&&) *this = (student&&) x;
        erase_links ();
        x.erase_links ();
        return *this;
      }

    friend class tree;
  private:
    void erase_links ()
      { left  = nullptr; right = nullptr; }
  };

class tree
  {
  private:
    tree_node * root = nullptr;
  public:
    tree () = default;
    tree (const tree& x) = delete;
    tree (tree&& x)
      {
        root = x.root; x.root = nullptr;
      }
    ~tree ()
      {
        delete_subtree (root);
```

```
      root = nullptr;
    }
  tree& operator= (const tree& x) = delete;
  tree& operator= (tree&& x)
    {
      if (this == &x)
        return *this;
      delete_subtree (root);
      root = x.root; x.root = nullptr;
      return *this;
    }
  void print (unsigned int r = 10, FILE *fp = stdout) const
    {
      print_subtree (root, 0, r, fp);
    }
  io_status read (FILE * fp = stdin, unsigned int max_read = -1);
private:
  static void delete_subtree (tree_node * curr)
    {
      if (curr == nullptr)
        return;
      delete_subtree (curr->left);
      delete_subtree (curr->right);
      delete curr;
    }
  static void print_subtree (tree_node * curr, int level, int r,
                             FILE *fp = stdout)
    {
      if (curr == nullptr || level > r)
        return;
      int spaces = level * 2;
      for (int i = 0; i < spaces; i++)
        printf (" ");
      curr->print (fp);
      print_subtree (curr->left, level + 1, r, fp);
      print_subtree (curr->right, level + 1, r, fp);
    }
  static void add_node_subtree (tree_node * curr, tree_node *x);
};
```

Все функции в задании являются членами класса "дерево".

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

   1) *r* – количество выводимых уровней в дереве,
   2) `filename` – имя файла, откуда надо прочитать дерево.

   Например, запуск

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

означает, что дерево надо прочитать из файла `a.txt`, выводить не более 4-х уровней дерева.

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

4. 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 `student` попадает в корень дерева, каждый новый объект типа `student` добавляется в левое поддерево, если он меньше текущего узла относительно операции сравнения `tree_node::operator<`, 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.

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

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

   где

   - `argv[0]` – первый аргумент командной строки (имя образа программы),
   - `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.

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

9. Поскольку дерево не изменяется всеми функциями из задач, кроме последней, то для всех задач надо сделать **одну функцию** `main`, в которой прочитать дерево, вывести указанное число уровней на экран, и вызвать все функции задач, выводя 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. Другими словами, после компиляции должен получиться **один исполняемый файл** `a.out`, а не несколько.

Задачи

1. Написать подпрограмму – член класса "дерево", возвращающую целое значение, равное количеству концевых элементов этого дерева.

2. Написать подпрограмму – член класса "дерево", возвращающую целое значение, равное длине наибольшей ветви этого дерева. Длина ветви измеряется в количестве элементов в ветви.

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

4. Написать подпрограмму – член класса "дерево", возвращающую целое значение, равное максимальной по модулю разности между глубинами левого и правого поддеревьев в узлах дерева. Глубина поддерева – это количество уровней в нем.

5. Написать подпрограмму – член класса "дерево", возвращающую целое значение, равное количеству элементов, имеющих ровно 1 потомка.

6. Написать подпрограмму – член класса "дерево", которая удаляет все узлы дерева, имеющие наименьшее значение поля `value` среди всех узлов дерева. Узел 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, имеющие наименьшее значение поля `value`, без учета потомков).