Требования к программам

1. В программе должны быть реализованы следующие структуры данных:

   - Контейнер данных объектов типа `student`:

```
#include <stdio.h>
#include <string.h>
#include <memory>

enum class io_status
{
  success,
  eof,
  format,
  memory,
  open,
  create,
};

class student
{
  private:
    std::unique_ptr<char []> name;
    int    value = 0;
  public:
    student () = default;
    student (const student& x) = delete;
    // По умолчанию: переместить все поля класса - подходит
    student (student&& x) = default;
    ~student () = default;
    student& operator= (const student& x) = delete;
    // По умолчанию: присвоить с перемещением все поля класса - подходит
    student& operator= (student&& x) = default;
    void print (FILE * fp = stdout) const
      {
        fprintf (fp, "%s %d\n", name.get (), value);
      }
    io_status read (FILE * fp = stdin)
      {
        const int LEN = 1234;
        char n[LEN];
        int v;

        if (fscanf (fp, "%s%d", n, &v) != 2)
          return io_status::format;
        erase ();
        return init (n, v);
      }
    int cmp (const student& x) const
      {
        // Используем переопределенную функцию сравнения с nullptr
        if (name == nullptr)
          {
            if (x.name != nullptr)
```

```
            return -1;
          return value - x.value;
        }
      if (x.name.get () == nullptr)
        return 1;
      int res = strcmp (name.get (), x.name.get ());
      if (res)
        return res;
      return value - x.value;
    }
  int operator<  (const student& x) const { return cmp (x) <  0; }
  int operator<= (const student& x) const { return cmp (x) <= 0; }
  int operator>  (const student& x) const { return cmp (x) >  0; }
  int operator>= (const student& x) const { return cmp (x) >= 0; }
  int operator== (const student& x) const { return cmp (x) == 0; }
  int operator!= (const student& x) const { return cmp (x) != 0; }

private:
  io_status init (const char * n, int v)
    {
      value = v;
      if (n != nullptr)
        {
          size_t len = strlen (n);
          name = std::make_unique<char []> (len + 1);
          if (name != nullptr)
            {
              for (size_t i = 0; i <= len; i++)
                name[i] = n[i];
            }
          else
            return io_status::memory;
        }
      return io_status::success;
    }
  void erase ()
    {
      value = 0;
      name.reset ();
    }
};
```

- AVL-дерево

```
template <class T> class avl_tree;
template <class T>
class avl_tree_node : public T
  {
  private:
    avl_tree_node * left   = nullptr;
    avl_tree_node * right  = nullptr;
    int balance = 0;
  public:
    avl_tree_node () = default;
    ...
```

```
    friend class avl_tree<T>;
  };
template <class T>
class avl_tree
  {
  private:
    avl_tree_node<T> * root = nullptr;
  public:
    avl_tree () = default;
    ...
    int read (FILE * fp = stdin);
    void print (int r, FILE *fp = stdout);
  };
```

2. Достаточно реализовать только функцию добавления элементов в дерево с сохранением его структуры. Эта функция вызывается при чтении элементов. Дерево удаляется в деструкторе целиком, поэтому функцию удаления элемента из дерева с сохранением его структуры можно не реализовывать.

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

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

   1) r – максимальное количество выводимых уровней в дереве,
   2) s – строка, параметр задачи,
   3) `filename` – имя файла, откуда надо прочитать дерево.

   Например, запуск

   ```
   ./a.out 4 "abc" a.txt
   ```

   означает, что AVL-дерево надо прочитать из файла `a.txt`, выводить не более 4-х уровней дерева, и вычислить результат задач для $s = abc$.

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. Все записи в файле различны (т.е. нет двух, у которых совпадают все поля). **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, то для всех задач надо сделать **одну функцию** `main`, в которой создается объект `avl_tree<student>`, вводится из указанного файла, выводится указанное число уровней на экран, вызываются все функции задач, выводятся результаты и время их работы; затем удаляется этот объект. После компиляции должен получиться один исполняемый файл `a.out`, а не несколько.

9. Каждая задача вызывается два раза: вначале для константной строки *s*, содержащей все буквы латинского алфавита (строчные и прописные), а затем для указанной в качестве аргумента строки *s*.

10. Схема функции `main`:

```
int main (int argc, char *argv[])
{
  // Ввод аргументов командной строки
  ...
  avl_tree<student> *a = new avl_tree<student>;
  // Работа с деревом avl_tree<student>
  ...
  delete a;
  return 0;
}
```

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

```
const char * s_all
  = "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ";
измеряем время t_all работы: int res_all = task_номер (s_all);
измеряем время t работы: int res = task_номер (s);
printf ("%s : Task = %d S = %s Result = %d Elapsed = %.2f\n",
        argv[0], task, s_all, res_all, t_all);
printf ("%s : Task = %d S = %s Result = %d Elapsed = %.2f\n",
        argv[0], task, s, res, t);
```

где

- `argv[0]` – первый аргумент командной строки (имя образа программы),
- `task` – номер задачи (1–5),
- `s` – параметр *s* командной строки,
- `res`, `res_all` – результат работы функции, реализующей решение этой задачи, соответственно для строк `s` и `s_all`,
- `t`, `t_all` – время работы функции, реализующей решение этой задачи, соответственно для строк `s` и `s_all`.

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

Задачи

1. Написать функцию – член класса "AVL-дерево", получающую в качестве аргумента строку *s*, и возвращающую целое значение, равное количеству концевых элементов этого дерева, в которых поле `name` состоит из символов строки *s*.

2. Написать функцию – член класса "AVL-дерево", получающую в качестве аргумента строку *s*, и возвращающую целое значение, равное максимальному количеству элементов в одной ветви этого дерева, в которых поле `name` состоит из символов строки *s*.

3. Написать функцию – член класса "AVL-дерево", получающую в качестве аргумента строку *s*, и возвращающую целое значение, равное количеству поддеревьев этого дерева, в которых поле `name` состоит из символов строки *s*.

4. Написать функцию – член класса "AVL-дерево", получающую в качестве аргумента строку *s*, и возвращающую целое значение, равное максимальному количеству элементов в одном уровне этого дерева, в которых поле `name` состоит из символов строки *s*.

5. Написать функцию – член класса "AVL-дерево", получающую в качестве аргумента строку *s*, и возвращающую целое значение, равное максимальной по модулю разности между количеством элементов, в которых поле `name` состоит из символов строки *s*, в левом поддереве и количеством таких элементов в правом поддереве.