Требования к программам

1. В программе должны быть реализованы следующие структуры данных:

   - Контейнер данных объектов типа `student`:

```
class student
  {
  private:
    char * name = nullptr;
    int    value = 0;
  public:
    student () = default;
    ...
    int read (FILE * fp);
    void print ();
    int operator< (const student& b);
  };
```

   - Красно-черное дерево (RB-дерево)

```
template <class T> class rb_tree;
template <class T>
class rb_tree_node : public T
  {
    enum colors
      {
        invalid,
        red,
        black,
      };
  private:
    rb_tree_node * left = nullptr;
    rb_tree_node * right = nullptr;
    colors color = invalid;
  public:
    rv_tree_node () = default;
    ...
    friend class rb_tree<T>;
  };
template <class T>
class rb_tree
  {
  private:
    rb_tree_node<T> * root = nullptr;
  public:
    rd_tree () = default;
    ...
    int read (const char * filename);
    void print (int r);
  };
```

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

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

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

   Например, запуск

   ```
   ./a.out 4 3 a.txt
   ```

   означает, что RB-дерево надо прочитать из файла `a.txt`, выводить не более 4-х уровней дерева, и вычислить результат задач для $k = 3$.

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

5. 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). RB-дерево заполняется как упорядоченное дерево. **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.

6. 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.

7. Поскольку дерево не изменяется функциями из задач, то для всех задач надо сделать **одну функцию** `main`, в которой создается объект `b_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.

8. Схема функции `main`:

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

9. 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 (1–5),
- `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 функцию – член класса "RB-дерево", получающую в качестве аргумента целое число k, и возвращающую целое значение, равное количеству элементов типа `student` в поддеревьях, имеющих не более k вершин.

2. Написать функцию – член класса "RB-дерево", получающую в качестве аргумента целое число k, и возвращающую целое значение, равное количеству элементов типа `student` в поддеревьях, имеющих не более k уровней.

3. Написать функцию – член класса "RB-дерево", получающую в качестве аргумента целое число k, и возвращающую целое значение, равное количеству элементов типа `student` в поддеревьях, имеющих не более k узлов в любом уровне.

4. Написать функцию – член класса "RB-дерево", получающую в качестве аргумента целое число k, и возвращающую целое значение, равное количеству элементов типа `student` в его k-м уровне.

5. Написать функцию – член класса "RB-дерево", получающую в качестве аргумента целое число k, и возвращающую целое значение, равное количеству элементов типа `student` во всех ветвях длины k, начиная с корня.