

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.
3. Аргументы командной строки для задач 1–4:

- 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.

4. Аргументы командной строки для задач 5–7:

- 1) r – количество выводимых уровней в дереве,
- 2) filename – имя файла, откуда надо прочитать дерево,
- 3) name – строка,
- 4) value – значение,

где поля `name` и `value` задают один объект типа `student`. Например, запуск

```
./a.out 4 a.txt Student 5
```

означает, что дерево надо прочитать из файла `a.txt`, выводить не более 4-х уровней дерева и создать элемент типа `student` с полями `{"Student", 5}`.

5. Задачи оцениваются независимо в двух группах: задачи 1–4 и задачи 5–7.
6. Класс "дерево" должен содержать функцию ввода дерева из указанного файла.
7. 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.

8. Решение задач должно быть оформлено в виде подпрограмм, находящихся в отдельном файле. Получать в этих подпрограммах дополнительную информацию извне через глобальные переменные, включаемые файлы и т.п. запрещается.
9. Для контроля структуры дерева на больших наборах данных требуется реализовать следующие функции-члены класса "дерево", возвращающие целое значение, равное
 - количеству элементов дерева: `int get_count_total () const;`
 - количеству концевых элементов дерева: `int get_count_leaf () const;`
 - количеству элементов, имеющих ровно 1 потомка: `int get_count_1 () const;`
 - количеству элементов, имеющих ровно 2 потомка: `int get_count_2 () const;`
 - длине наибольшей ветви этого дерева (длина ветви измеряется в количестве элементов в ветви): `int get_height () const;`
 - максимальному количеству элементов, находящихся в одном уровне этого дерева: `int get_width () const;`
 - максимальной по модулю разности между глубинами левого и правого поддеревьев в узлах дерева (глубина поддерева – это количество уровней в нем): `int get_balance () const;`
10. Класс "дерево" должен содержать подпрограмму `void print (int r, FILE *fp) const` 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.
11. Программа должна выводить на экран время, затраченное на решение.

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

```
tree a;
// ... Initialize the tree ...
printf ("Original tree:\n");
a.print (r);
int T = a.get_count_total ();
int L = a.get_count_leaf ();
int C1 = a.get_count_1 ();
int C2 = a.get_count_2 ();
int H = a.get_height ();
int W = a.get_width ();
int B = a.get_balance ();
printf ("%s : Task = %d T = %d L = %d C1 = %d C2 = %d H = %d W = %d B = %d\n", argv[0], task, T, L, C1, C2, H, W, B);
double t = clock ();
solve (); // solution
t = (clock() - t)/CLOCKS_PER_SEC;
printf ("Modified tree:\n");
a.print (r);
T = a.get_count_total ();
L = a.get_count_leaf ();
C1 = a.get_count_1 ();
C2 = a.get_count_2 ();
H = a.get_height ();
W = a.get_width ();
B = a.get_balance ();
printf ("%s : Task = %d T = %d L = %d C1 = %d C2 = %d H = %d W = %d B = %d Elapsed = %.2f\n", argv[0], task, T, L, C1, C2, H, W, B, t);
```

где

- `argv[0]` – первый аргумент командной строки (имя образа программы),
- `r` – количество выводимых уровней в дереве (аргумент командной строки),
- `task` – номер задачи (1–5),
- `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", преобразовывающую дерево так, чтобы в каждой вершине стоял максимальный в своем поддереве элемент.
2. Написать подпрограмму – член класса "дерево", преобразовывающую дерево так, чтобы в каждом уровне элементы шли в порядке возрастания поля `value`, методом "нахождения минимума".
3. Написать подпрограмму – член класса "дерево", преобразовывающую дерево по следующему правилу: начиная с корня, для каждого узла, у которого нет левого потомка (назовем этот узел *A*) и есть правый потомок (назовем его *B*), не являющийся концевым, найти в поддереве с вершиной *B* "самый левый" концевой элемент (назовем его *C* – это элемент, для которого при движении по цепочке от узла *C* к родительскому узлу *B* переход идет всегда от левого потомка к родителю),

и если такой элемент C существует, то отсоединить этот узел C , поставить его на место узла A , сделав у C родителем родителя узла A (если такой есть), правым потомком узел B , а левым потомком – узел A , который в свою очередь становится концевым с родителем – узлом C .

4. Написать подпрограмму – член класса "дерево", преобразовывающую дерево по следующему правилу: начиная с корня, для каждого узла, у которого нет правого потомка (назовем этот узел A) и есть левый потомок (назовем его B), не являющийся концевым, найти в поддереве с вершиной B "самый правый" концевой элемент (назовем его C – это элемент, для которого при движении по цепочке от узла C к родительскому узлу B переход идет всегда от правого потомка к родителю), и если такой элемент C существует, то отсоединить этот узел C , поставить его на место узла A , сделав у C родителем родителя узла A (если такой есть), левым потомком узел B , а правым потомком – узел A , который в свою очередь становится концевым с родителем – узлом C .
5. Написать подпрограмму – член класса "дерево", получающую в качестве аргумента константную ссылку на элемент типа `student`, в котором обозначим строку `name` через s , а число `value` через k . Подпрограмма должна удалить все поддеревья глубиной не менее k , такие, что в поле `name` каждого узла поддерева входит строка s .
6. Написать подпрограмму – член класса "дерево", получающую в качестве аргумента константную ссылку на элемент типа `student`, в котором обозначим строку `name` через s , а число `value` через k . Подпрограмма должна удалить все поддеревья, имеющие ветвь длиной не менее k , такие, что в поле `name` каждого узла этой ветви входит строка s .
7. Написать подпрограмму – член класса "дерево", получающую в качестве аргумента константную ссылку на элемент типа `student`, в котором обозначим строку `name` через s , а число `value` через k . Подпрограмма должна удалить все поддеревья, начинающиеся с элемента уровня, состоящего из не менее k подряд идущих элементов уровня, в каждый из которых в поле `name` входит строка s .