

Практика #4.

Бинарное дерево поиска.

1. Реализовать структуру данных бинарное дерево поиска. Для этого создать шаблон класса `BinarySearchTree`.

1.1 Для описания узла дерева используйте тип `Node`, в котором поля:

- `key_` - значение ключа узла,
- `left_` - указатель на левое поддерево,
- `right_` - указатель на правое поддерево,
- `p_` - указатель на родителя (может не использоваться).

Тип `Node` может использоваться только в классе `BinarySearchTree`.

Класс `BinarySearchTree` должен содержать поле `root_` - указатель на корневой узел.

1.2 В классе должны быть:

- конструктор по умолчанию, создающий пустое дерево,
- конструктор перемещения,
- оператор перемещающего присваивания,
- деструктор.

Конструктор копирования и оператор присваивания (с копированием) должны быть запрещены.

1.3 В классе должны быть методы:

1. поиска по ключу (итеративный)
2. вставки нового элемента в дерево (итеративный)
3. удаления элемента из дерева (итеративный)
4. печати строкового изображения дерева
5. определения количества узлов дерева (рекурсивный)
6. определения высоты дерева (рекурсивный)
7. инфиксного обхода дерева (итеративный)
8. инфиксного обхода дерева (рекурсивный)

Два набора методов:

1. `private` для работы с узлами – для разработчика класса
2. `public` работы со значениями (ключами) – для пользователя

Методы могут быть перегруженными, т. е. можно использовать одно и то же имя для `private` и `public` методов.

2. Пример реализации метода определения количества узлов

```
// private: Рекурсивная функция определения количества узлов дерева
size_t getCount (Node *node) const
{
    if (node == nullptr) {
        return 0;
    }
```

```

    }
    return (1 + getCount(node->left_) + getCount(node->right_));
}
//
// public: Определение количества узлов дерева
size_t getCount () const
{
    return getCount(this->root);
};

```

3. Написать функции для отладки методов.
4. В реализации использовать приведенные ниже заготовки:

Файл BinarySearchTree.h

```

#ifndef _BINARY_SEARCH_TREE_H
#define _BINARY_SEARCH_TREE_H

template <class T>
class BinarySearchTree
{
public:
    BinarySearchTree(); // "по умолчанию" создает пустое дерево
    BinarySearchTree(const BinarySearchTree<T> & scr) = delete;
    BinarySearchTree(BinarySearchTree<T>&& scr);
    BinarySearchTree <T>& operator= (const BinarySearchTree <T>& src) = delete;
    BinarySearchTree <T>& operator= (BinarySearchTree <T>&& src);
    virtual ~BinarySearchTree();

    // 1.1 Функция поиска по ключу в бинарном дереве поиска
    bool iterativeSearch(const T& key) const;

    // 2 Вставка нового элемента в дерево: true, если элемент добавлен;
    // false, если элемент уже был
    bool insert(const T& key);

    // 3.1 Удаление элемента из дерева, не нарушающее порядка элементов
    // true, если элемент удален; false, если элемента не было
    bool deleteKey(const T& key);

    // 4.1 Печать строкового изображения дерева в выходной поток out
    void print(std::ostream& out) const;

    // 5.1 Определение количества узлов дерева
    int getCount() const;

    // 6.1 Определение высоты дерева
    int getHeight() const;

    // 7 Инфиксный обход дерева (итеративный)
    void iterativeInorderWalk() const;

    // 8.1 Инфиксный обход дерева (рекурсивный)
    void inorderWalk() const;

```

private:

```
// 1.2 Функция поиска адреса узла по ключу в бинарном дереве поиска
Node<T>* iterativeSearchNode(const T& key) const;
```

```
// 4.2 Рекурсивная функция для вывода изображения дерева в выходной поток
void printNode(std::ostream& out, Node<T>* root) const;
```

```
// 5.2 Рекурсивная функция определения количества узлов дерева
int getCount(const Node<T>* node) const;
```

```
// 6.2 Рекурсивная функция определения высоты дерева
int getHeight(const Node<T>* node) const;
```

```
// 8.2 Рекурсивная функция для организации обхода узлов дерева.
void inorderWalk(Node<T>* node) const;
```

template <class T>

```
struct Node {
    T key_; // значение ключа, содержащееся в узле
    Node<T> *left_; // указатель на левое поддерево
    Node<T> *right_; // указатель на правое поддерево
    Node<T> *p_; // указатель на родителя !!! не используется
    // Конструктор узла
    Node(T key, Node *left = nullptr, Node *right= nullptr, Node *p =nullptr):
        key_(key), left_(left), right_(right), p_(p)
    { }
};
```

Node<T> *root_; // Указатель на корневой узел

```
}; // конец шаблона класса TreeBinarySearchTree
```