Лабораторная работа №6

Технология разработки алгоритмов решения инженерных задач

Тема: Деревья алгоритмы на деревьях.

цель: по варианту определить задачи; сформулировать упрощения задачи, описать принцип работы алгоритма как последовательное упрощение задачи; блок-схему рекуррентного алгоритма; доказать гарантированность достижения развязку.

Задание

1. Определите номер варианта. С указанной таблицы по первым буквам фамилии и имя определите две цифры. Вычислите номер своего варианта:

№ (буква с фамилии) * 6 + № (буква из имени) = последняя цифра № вашего варианта

А	Б	В	Г	Д	Е	Есть	Ж	С	И
0	1	2	3	4	5	0	1	2	3
Й	И	К	Л	М	Н	В	П	Р	С
4	5	0	1	2	3	4	5	0	1
Т	В	Ф	Х	Ц	Ч	Ш Щ Ю			Я
2	3	4	5	0	1	2	3	4	5

2. Запишите структуру программы для обеспечения работы с бинарным деревом, которое содержит в качестве элементов целые числа.

3. Опишите действия по пунктам, которые нужно выполнить для добавления элемента к дереву так, чтобы дерево оставалось упорядоченным.

4. В каком порядке нужно добавить к упорядоченному дерева числа 1,2, ..., 15 чтобы оно было уравновешенным?

5. вправе ошибки в реализации дерева на примере и заполните его числами от 1 до 15 так, чтобы дерево было уравновешенным. Напишите и настройте программу.

6. По определенным вариантом с последней цифры выберите свою задачу:

№ вар.	Задача согласно варианту
0	**Выходные данные:** Уравновешенное упорядоченное дерево с числами 1,2, ..., 15. **Выходные данные:** Числа по порядку обхода методом A (см. Пример). 1 **Выходные данные:** Уравновешенное
	упорядоченное дерево с числами 1,2, ..., 15. **Выходные данные:** Числа по порядку обхода методом B (см. Пример). 2 **Выходные данные:** Уравновешенное
	упорядоченное дерево с числами 1,2, ..., 15. **Выходные данные:** Числа по порядку обхода методом C (см. Пример). 3 **Выходные данные:** Уравновешенное
	упорядоченное дерево с числами 1,2, ..., 15. **Выходные данные:** Числа по порядку обхода методом D (см. Пример). 4 **Выходные данные:** Уравновешенное
	упорядоченное дерево с числами 1,2, ..., 15. **Выходные данные:** Числа по порядку обхода методом E (см. Пример). 5 **Выходные данные:** Уравновешенное
	упорядоченное дерево с числами 1,2, ..., 15.

	Выходные данные: Числа по порядку обхода методом F (см. Пример). 6 **Выходные данные:** Уравновешенное
упорядоченное дерево с числами 1,2, ..., 15.	**Выходные данные:** Числа по порядку обхода методом A (см. Пример). 7 **Выходные данные:** Уравновешенное
упорядоченное дерево с числами 1,2, ..., 15.	**Выходные данные:** Числа по порядку обхода методом B (см. Пример). 8 **Выходные данные:** Уравновешенное
упорядоченное дерево с числами 1,2, ..., 15.	**Выходные данные:** Числа по порядку обхода методом C (см. Пример). 9 **Выходные данные:** Уравновешенное
упорядоченное дерево с числами 1,2, ..., 15.	**Выходные данные:** Числа по порядку обхода методом D (см. Пример). Запишите результат работы

программы. Методы выведут номера по возрастанию, а какие по убыванию?

7. Словами описать процесс добавления к дереву числа 17. Будет ли новое дерево уравновешенным?

8. Повторить опыты п.6 для дерева, заполнены числами в таком порядке, нарисуйте полученное дерево: N = (№ варианта + 3); N, N-1, N-2, ..., 1, N + 1, N + 2, ..., 15

(Пример. Номер варианта 4. N = 4 + 3 = 7. К дереву добавляем: 7,6,5,4,3,2,1,8,9,10,11,12,13,14,15)

9. Оценить сложность алгоритма.

10. Записать выводы о проделанной работе.

11. Ответить на контрольные вопросы (в день выполнения работы устно, при пересдаче или к сдаче и др. - письменно).

12. Дополнительное задание. Дополнить класс дерева функцией рисования дерева на экране монитора. Копию экрана добавить в отчет. (+4 балла)

Вспомогательная информация. Реализация бинарного дерева C ++

```
class CElement {public:

  int E; // Элемент, который хранится в дереве CElement * Left; // Переход
влево-вниз по дереву CElement * Right; // Переход вправо-вниз по дереву
CElement (int i) // Конструктор ~ CElement (); // Деструктор};



Celement :: CElement (int i = 0) // Конструктор {E = i; // Значение

элемента дерева
  Left = null; // Элемент новый, поэтому ниже пусто Right = NULL; // Элемент
новый, поэтому ниже пусто}


CElement :: ~ CElement () // Деструктор
{If (Left! = NUll) delete Left; // Если снизу что-то есть, то

  if (Right! = null) delete Right; // это тоже надо уничтожить}


// Класс управления деревом class
CTree
```

```
{Private:

        CElement Root; public:

        CTree (); // Конструктор дерева ~ CTree (); //
        Деструктор дерева
        addElement (int A) // Добавление элемента
                                                // с сохранением упорядоченности
        void clearTree (); // Удаление все элементы из дерева void printTreeA (CElement * Base =
       NULL); // Обход дерева void printTreeB (CElement * Base = NULL); // различным порядком
       void printTreeC (CElement * Base = NULL); void printTreeD (CElement * Base = NULL); void
       printTreeE (CElement * Base = NULL); void printTreeF (CElement * Base = NULL); };




CTree :: CTree () // Конструктор дерева {Root = NULL;


} CTree :: ~ CTree () // Деструктор дерева {delete Root;



} Void CTree :: addElement (int A) // Добавление элемента {

                                                // с сохранением упорядоченности
    if (Root == Null)
    {// Дерево пустое, добавим элемент как корневой.
        Root = new CElement (A); return; } // Дерево уже есть, нужно правильно спуститься
     по нему CElement * Now = null; // Настоящее положение в дереве CElement * Next =
     Root; // Следующее положение в дереве



                                                // начнем с корня
    do { Now = Next; // Переходим на следующий элемент вниз

                                    // (в начале в корень)
        if (Now-> E <A) // Если наше число больше, то {// дальше нам нужно
        направо по дереву
            Next = Now-> Right; } else

        {// иначе - спускаемся налево, потому вставляем меньше элемент
            Next = Now-> Left; }

    } While (Next! = NULL); // Повторяем, пока следующий элемент
                                                  // не станет пустым - мы спустились
    if (Now-> E <A) // Теперь последнее решение о добавлении элемента {// с правой стороны

        Now-> Right = new CElement (A); } else

    {// или слева
```

```
        Now-> Left = new CElement (A); }

} Void CTree :: clearTree () // Удаление все элементы из дерева {delete Root; // Все


остальное вилучиться автоматически
                                // деструктором элемента
} Void CTree :: printTreeA (CElement * Base = NULL) // Обход дерева {CElement * Now = Base;



    if (Now == null) Now = Root; if (Now == null)
    return; cout << Now-> E << endl;

    if (Now-> Left! = NULL) printTreeA (Now-> Left) if (Now-> Right! = NULL)

 printTreeA (Now-> Right) } Void CTree :: printTreeB (CElement * Base = NULL)

 {CElement * Now = Base;


    if (Now == null) Now = Root; if (Now == null)
    return;
    if (Now-> Left! = NULL) printTreeA (Now-> Left) cout << Now-> E << endl;

    if (Now-> Right! = NULL) printTreeA (Now-> Right) } Void CTree :: printTreeC

(CElement * Base = NULL) {CElement * Now = Base;


    if (Now == null) Now = Root; if (Now == null)
    return;
    if (Now-> Left! = NULL) printTreeA (Now-> Left) if (Now-> Right! = NULL)
    printTreeA (Now-> Right) cout << Now-> E << endl;

} Void CTree :: printTreeD (CElement * Base = NULL) {CElement * Now


= Base;
    if (Now == null) Now = Root; if (Now == null)
    return; cout << Now-> E << endl;

    if (Now-> Right! = NULL) printTreeA (Now-> Right) if (Now-> Left! = NULL)

 printTreeA (Now-> Left) } Void CTree :: printTreeE (CElement * Base = NULL)

 {CElement * Now = Base;


    if (Now == null) Now = Root; if (Now == null)
    return;
    if (Now-> Right! = NULL) printTreeA (Now-> Right) cout << Now-> E << endl;

    if (Now-> Left! = NULL) printTreeA (Now-> Left) }
```

```
void CTree :: printTreeF (CElement * Base = NULL) {CElement * Now =

Base;
    if (Now == null) Now = Root; if (Now == null)
    return;
    if (Now-> Right! = NULL) printTreeA (Now-> Right) if (Now-> Left! = NULL)
  printTreeA (Now-> Left) cout << Now-> E << endl; }
```