МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ОТЧЕТ

по лабораторной работе №3

по дисциплине «АиСД»

Тема: Деревья

Студент гр. 9382 Пя С.

Преподаватель Фирсов М.А.

Санкт-Петербург
2020

Цель работы.

Познакомиться со структурой деревьев. Научиться создавать бинарные деревья, реализовывая их через динамическую (связанную) память (на базе указателей), и пользоваться ими. Закрепить их реализацию с помощью рекурсии на примере языка C++, освоить применение структур и классов, получить навыки работы с template c++. Выполнить работу в соответствии с заданием.

Основные теоретические положения.

Дерево – конечное множество T, состоящее из одного или более узлов, таких, что

а) имеется один специально обозначенный узел, называемый корнем данного дерева;

б) остальные узлы (исключая корень) содержатся в m >= 0 попарно не пересекающихся множествах T1, T2, ..., Tm, каждое из которых, в свою очередь, является деревом. Деревья T1, T2, ..., Tm называются поддеревьями данного дерева.

Лес – это множество (обычно упорядоченное), состоящее из некоторого (быть может, равного нулю) числа непересекающихся деревьев. Используя понятие леса, пункт *б* в определении дерева можно было бы сформулировать так: *узлы дерева, за исключением корня, образуют лес*.

КЛП – префиксная запись

ЛКП – инфиксная запись

ЛПК – постфиксная запись

Задание

Вариант №9д.

Рассматриваются бинарные деревья с элементами типа Elem (в качестве Elem использовать char). Заданы перечисления узлов некоторого дерева b в порядке КЛП и ЛКП. Требуется:

- восстановить дерево b и вывести его изображение;

- перечислить узлы дерева b в порядке ЛПК.

Ход работы.

1) Разработан алгоритм:

На вход подаются два выражения, являющиеся разными записями одного и того же бинарного дерева (ЛКП и КЛП). Используя теоретические знания о том, что во втором выражении первыми узлами записываются корни поддеревьев, находим эти корни в первом выражении, они будут в не с краю, потому что в ЛКП записи сначала записываются левые, потом корни, после правые поддеревья. Затем мы делим первое выражение на два, они будут образовывать два поддерева и будут являться левым и правым поддеревом соответственно нашего главного дерева. Затем таким образом рекурсивно составляем из выражений бинарное дерево. Были устранены все утечки памяти, написана реакция на не открытие файла и на некорректные данные.

Предусмотрен механизм простейшего взаимодействия с пользователем, позволяющий понять алгоритм исполнения программы, с помощью вывода сообщений. Также был предусмотрен ввод данных с клавиатуры.

2) Использованы функции, методы, классы и структуры:

1. main

Сигнатура: int main().

Назначение: является основной функцией и телом программы.

Описание аргументов: без аргументов.

Возвращаемое значение: Функция возвращает 0.

2. BinaryTree

Назначение: класс дерева, в котором хранятся методы для создания дерева и его отображения.

Описание содержимого кроме методов: приватная структура struct Node, содержащая Elem data (использован шаблон template c++, заменяющийся на char) – хранит содержимое узла; указатели левый и правый struct Node* left, struct Node* right, которые ссылаются на левое и правое поддерево соответственно. Также есть конструктор Node(), инициализирующий указатели nullptr-ом. Структура описывает узел, нужна для выполнения рекурсивного алгоритма дерева. Из public две переменных: int i – счетчик для КЛП записи, чтобы по очереди брать узлы в строке, и node* tree, хранящее дерево, к которому можно обратиться. Далее идут методы класса:

3. createBinaryTree

Сигнатура: node* createBinaryTree(char LKP[], char KLP[]).

Назначение: приватный метод: создает бинарное дерево, состоящее из структур.

Описание аргументов: строки LKP и KLP, являющимися формами записи узлов дерева.

Возвращаемое значение: указатель на основной корень дерева типа node.

4. BinaryTree

Сигнатура: BinaryTree(char LKP[], char KLP[]).

Назначение: Публичный конструктор, инициализирующий переменные. В нем создается дерево.

Описание аргументов: строки LKP и KLP, являющимися формами записи узлов дерева.

5. printInorder

Сигнатура: void printInorder(node* node, int n, char* shift).

Назначение: составляет схематическое изображение дерева.

Возвращаемое значение: метод ничего не возвращает.

6. printLPK

Сигнатура: static void printLPK(node* node).

Назначение: выводит постфиксную запись ЛПК дерева.

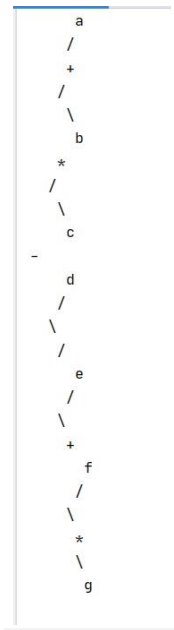
Описание аргументов: указатель на узел основного корня дерева типа node.

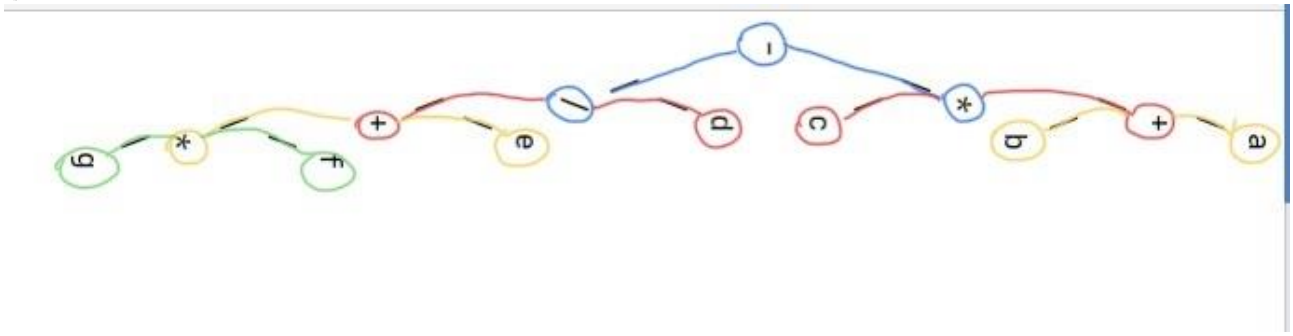
Возвращаемое значение: метод ничего не возвращает.

Реализация рекурсивного метода:

На вход подаются два выражения, являющимися ЛКП и КЛП записями. Затем проверяются данные на корректность. Создается узел дерева, куда записывается следующее значение в выражении КЛП. После значение ищется в выражении ЛКП, и это выражение делится на две части, которые отдаются в следующие методы рекурсии, пока в выражениях есть значения. После возвращается все дерево.

Пример работы программы.





Входные данные	Выходные данные
a+b*c-d/e+f*g - *+abc/d+e*fg	Do you want to enter data(0) or read it from file(write number of file)? 3 Data is: a+b*c-d/e+f*g -*+abc/d+e*fg a / + / \ b * /

	\ c - d / \ / e / \ + f / \ * \ g LPK is a b + c * d e f g * + / -

Тестирование.

Результаты тестирования представлены в табл. 1.

Таблица 1 – Результаты тестирования

№ п/п	Входные данные	Выходные данные	Комментарии
1.	425136 124536	Do you want to enter data(0) or read it from file(write number of file)? 1 Data is: 425136 124536 4 / 2 / \ 5	Проверка на корректность работы программы

		1 \ 3 6 / LPK is 4 5 2 6 3 1	
2.	1+4*9-5 *+14-95	Do you want to enter data(0) or read it from file(write number of file)? 2 Data is: 1+4*9-5 *+14-95 1 / + / \ 4 * 9 / \ - \ 5 LPK is 1 4 + 9 5 - *	Проверка на корректность работы
4.		Do you want to enter data(0) or read it from file(write number of file)? 4 Data is: Error Data	Проверка на корректность работы с пустыми строками
5.	14235 12345	Do you want to enter data(0) or read it from file(write number of file)? 5 Data is: 14235 12345 Error Data	Проверка на корректность работы с неверными данными

Выводы.

В ходе работы была освоена реализация бинарного дерева на основе рекурсии и структур, отработано понимание его применения, и отработаны навыки письма в C++.

Код программы можно найти в приложении А.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.cpp

```
 /*#define _CRTDBG_MAP_ALLOC
#include <crtdbg.h>
#define DEBUG_NEW new(_NORMAL_BLOCK, __FILE__, __LINE__)
#define new DEBUG_NEW*/
#include <iostream>
#include <fstream>
#include <cstring>
using namespace std;
template <typename Elem>// Elem - параметр шаблона класса, в качестве него
используем char
class BinaryTree {
private:
    typedef struct Node{//структура, описывающая содержимое узла
    Elem data;//данные, хранящиеся в узле
    struct Node* left;//левое поддерево
    struct Node* right;//правое поддерево
    Node() {
        left = nullptr;
        right = nullptr;
    }
} node;
    node* createBinaryTree(char LKP[], char KLP[]) {//рекурсивный метод для
создания дерева
        if (strlen (KLP) <= i) {//в случае некорректных данных выводится
сообщение об ошибке
            cout << "Error Data";
            return nullptr;
        }
        node* binaryTree = new node();
        binaryTree->data = KLP[i++];//содержимое узла
        char* rshift = strchr(LKP, binaryTree->data);
        if (rshift == nullptr) {//в случае некорректных данных выводится
сообщение об ошибке
            i = strlen(KLP) + 1;
            return nullptr;
        }
        char* lshift = new char[100]();
        strncat(lshift, LKP, rshift - LKP);
        strcat(lshift, "\0");
        if (strlen(lshift) != 0)
            binaryTree->left = createBinaryTree(lshift, KLP);//передаем левую
часть строки, из которой формируется поддерево
        if (strlen(rshift) != 1)//так как в правой части хранится искомый узел,
то здесь рекурсивный цикл прекращается на количестве элементов в правой части,
```

```
равном 1
            binaryTree->right = createBinaryTree(rshift + 1, KLP);//передаем
правую часть строки, из которой формируется поддерево
        delete[] lshift;
        return binaryTree;
    }
public:
    int i;//счетчик для KLP
    node* tree;//бинарное дерево, созданно для многократного безопасного
обращения к одному и тому же дереву
    BinaryTree(char LKP[], char KLP[]) {
        i = 0;
        tree = createBinaryTree(LKP, KLP);//создание дерева
    }
    void printInorder(node* node, int n, char* shift) {//вывод дерева
        n++;//номер уровня
        char k[10];
        char* _k_ = new char[10]();//строка, в которой будет хранится номер
уровня
        _k_[0] = ' ';
        if (node == NULL) {//завершение рекурсивного цикла
            delete[] _k_;
            return;
        }
        printInorder(node->left, n, shift);//вход в самую левую ветку
        itoa (n, k, 10);//перевод номера в строку
        strcat(_k_, k);
        strcat(_k_, " ");
        char* t = strstr(shift,_k_);//поиск номера в строке из номеров
        delete[] _k_;
        if (!(n == 1) && t) {//в случае нахождения текущего номера в строке
номеров знаем, что узел находится левее в бинарном дереве
            for (int i = 0; i < n + 1; i++)//сдвигаем узел до его уровня
                cout << " ";
            cout << node->data << "\n";
            char* tshift = new char[50]();
            for (int i = 0; i < n; i++)//рисуем ветку
                cout << " ";
            strncat(tshift, shift, t - shift);//убираем номер из строки номеров
            strcat(tshift, t + 2);
            delete[] shift;
            shift = new char[50]();
            strcat(shift, tshift);
            delete[] tshift;
            cout << "/\n";//рисуем ветку
        } else if (n != 1 && !t) {//в случае отсутствия текущего номера в строке
номеров знаем, что узел находится правее в бинарном дереве
            for (int i = 0; i < n; i++)
                cout << " ";
            cout << "\\\n";//рисуем ветку
            for (int i = 0; i < n + 1; i++)
                cout << " ";//сдвигаем узел до его уровня
            cout << node->data << "\n";
            itoa(n, k, 10);//добавляем номер обратно в строку номеров
            strcat(shift, k);
            strcat(shift, " ");
        } else//если номер 1, то это вершина дерева, его корень
            cout << node->data << "\n";
        printInorder(node->right, n, shift);//после самого левого прохода идем
вправый
    }

    static void printLPK(node* node) {//вывод постфиксной записи (ЛПК)
        if (node != nullptr) {
```

```
            printLPK(node->left);//сначала идем в левые поддеревья
            printLPK(node->right);//после в правые поддеревья
            cout << node->data << " ";//пишем узел
        }
    }
};

    int main() {
        auto* in = new char[30]();//строка для инфиксной записи ЛКП
        auto* pre = new char[30]();//строка для префиксной записи КЛП
        char x;//переменная для выбора ввода пользователем
        cout << "Do you want to enter data(0) or read it from file(write number
of file)?\n";
        cin >> x;
        if (x == 48) {//так как на вход принимается символ, то номер нуля = 48,
это работает, пока количество тестов не привышает 9
            cin >> in >> pre;//считывание строк из консоли
        }
        else {
            char *file = new char[50]();
            strcat(file, ".//Tests//");
            file[strlen(file)] = x;
            strcat(file, ".txt");
            fstream fin(file);//файл для считывания выражений
            if (!fin.is_open())
                throw ("File cannot be opened");
            fin >> in >> pre;//считывание строк из файла
            delete[] file;
        }
        cout << "Data is:\n" << in << "\n" << pre << "\n";
        BinaryTree<char> tree(in, pre);//создание экземпляра класса BinaryTree
        if (tree.tree == nullptr || strlen (pre) < tree.i)
            return -1;
        delete[] in;
        char* shift = new char[50]();//строка, в которой будут находиться номера
уровней узлов для вывода дерева
        shift[0] = ' ';
        char k[10];
        for (int i = 1;i <strlen(pre) + 1;i++) {//запись номеров в строку, они
не будут превышать количество узлов
            itoa(i, k, 10);
            strcat(shift, k);
            strcat(shift, " ");
        }
        shift[strlen(shift)] = '\0';
        tree.printInorder(tree.tree, 0, shift);//вывод дерева
        cout << "\nLPK is\n";
        tree.printLPK(tree.tree);//вывод постфикной записи
        delete[] pre;
 delete[] shift;
/*if (_CrtDumpMemoryLeaks())
     printf("\nMemory leack deteckted\n");
 else
     printf("\nMemory is not leack deteckted\n");*/
        return 0;
    }
```