

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

ОТЧЕТ
по лабораторной работе №3
по дисциплине «Алгоритмы и структуры данных»
Тема: Бинарные деревья

Студент гр. 9304

Борисовский В.Ю.

Преподаватель

Филатов Ар.Ю.

Санкт-Петербург

2020

Цель работы.

Ознакомиться с понятием бинарного дерева. Реализовать бинарное дерево для решения задания на языке программирования C++.

Задание.

Вариант 6.

Задано бинарное дерево b типа BT с произвольным типом элементов. Используя очередь, напечатать все элементы дерева b по уровням: сначала - из корня дерева, затем (слева направо) - из узлов, сыновних по отношению к корню, затем (также слева направо) - из узлов, сыновних по отношению к этим узлам, и т. д.

Выполнение работы.

1) Сперва создали класс `bin_tree_node`, данный класс предназначен для хранения узла бинарного дерева, он имеет поля:

`std::shared_ptr<bin_tree_node<T>>` `left` - левое поддерево.

`std::shared_ptr<bin_tree_node<T>>` `right` - правое поддерево.

`T data` - элемент шаблонного типа.

2) Затем был написан чекер строки, который проверяет ее на валидность. Чекер реализован в двух функциях - `bool first_checker(std::string &str, int &index)`, `bool second_checker(std::string &str)`. Первый чекер следит за тем, чтобы в случае когда выбрана опция «с» в конце строки в строке были только символы или же целые числа, если выбрана опция «i» (для определения опции также были написаны соответствующие чекеры). Первый чекер следит также за тем, что у корня дерева должно быть два наследника и не больше. Второй же чекер выполняет проверку на то, что после последней закрывающей скобки не следует лишних символов.

3) Затем реализовали класс `bin_tree()`. В конструкторе данного класса в начале проверяется чекером строка переданная для создания дерева, если чекер вернул `true`, значит все прошло успешно и для поля `std::shared_ptr<bin_tree_node<T>>` `head` вызывается функция `create_bin_tree(str, index)`, которая работает примерно как чекер, отличие в том, что в ней создаются узлы `bin_tree_node` и индексация происходит несколько иначе. Зная, что строка уже валидна эта функция легко проходится по данной строке и создает из нее дерево.

4) Далее выполнили условие задания своего варианта. Реализовали обход бинарного дерева в ширину, который как раз и обеспечивает вывод всех элементов бинарного дерева по уровням с использованием очереди. Алгоритм данного обхода крайне прост: создается очередь, в нее кладется корень дерева. Затем пока очередь не станет пустой начинаем извлекать из нее первый элемент, и класть левого и правого наследника извлеченного элемента. Метод `show_tree()`.

5) После этого реализовали функции LKP обхода, вставки и удаления элемента, требующиеся в условия лаб.работы.

6) В заключение была реализована функция `main()`, в ней из аргументов командой строки принимается строка, проверяется чекером опций «с» или же «i» затем создается объект класса `bin_tree` и для него вызывается метод `show_tree()`, иначе выводится сообщение о том, что строка неверна.

Тестирование.

Запуск программы начинается с ввода команды “make”, что приведёт к компиляции программы и созданию исполняемого файла `lab3`. Запуск программы производится командой `./lab3` и последующим вводом строки, содержащей логическое выражение. Тестирование производится с помощью скрипта `test_skript.py`. Запуск скрипта производится командой «python3 `test_skript.py`» в директории `lab3`. Результаты тестирования представлены в приложении Б.

Выводы.

Было изучено понятие бинарного дерева. Было реализовано бинарное дерево с помощью языка программирования C++, с использованием умных указателей `std::shared_ptr`.

Была реализована программа, печатающая все элементы дерева `b` по уровням: сначала - из корня дерева, затем (слева направо) - из узлов,

сыновних по отношению к корню, затем (также слева направо) - из узлов, сыновних по отношению к этим узлам, и т. д.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.cpp

```
#include <iostream>
#include <string>
#include <memory>
#include <queue>
#include <cstdlib>

//проверка на работу с char
bool char_checker(std::string &str){
    if (str[str.length() - 1] == 'c'){
        return true;
    } else {
        return false;
    }
}

//проверка на работу с int
bool int_checker(std::string &str){
    if (str[str.length() - 1] == 'i'){
        return true;
    } else {
        return false;
    }
}

//валидация строка до последней закрывающей скобки
bool first_checker(std::string &str, int &index){
    if (str[index] != '('){
        return false;
    }
}
```

```

    }
    index++;
    if (char_checker(str)) {
        if (str[index] != '(' && str[index] != ')') {
            index++;
        } else if (str[index] == ')') {
            return true;
        } else if (str[index] == '(') {
            return false;
        }
        if (str[index] == ')') {
            return true;
        }
    } else if (int_checker(str)){
        if (str[index] != '(' && str[index] != ')') {
            std::string check_num = "";
            while (str[index] != '(' && str[index] !=
'')) {

                check_num += str[index];
                index++;
            }
            if (!check_num.empty()) {
                char *endptr;

                const char *c_string =
check_num.c_str();

                strtol(c_string, &endptr, 10);
                if (*endptr) {
                    return false;
                }
            } else {
                return false;
            }
        } else if (str[index] == ')') {

```

```

        return true;
    } else if (str[index] == '(') {
        return false;
    }
    if (str[index] == ')') {
        return true;
    }
}

if (!first_checker(str, index)){
    return false;
}
index++;
if (!first_checker(str, index)){
    return false;
}
index++;

if (str[index] == ')'){
    return true;
} else {
    return false;
}
}

```

```

//проверка на то, что строка кончается валидно
bool second_checker(std::string &str){
    int index = 0;
    if(first_checker(str, index) && index + 2 ==
str.length()){
        return true;
    } else {

```



```

        return false;
    }
}

```

//структура узла бинарного дерева

```

template <typename T>
class bin_tree_node{
public:
    std::shared_ptr<bin_tree_node<T>> left;
    std::shared_ptr<bin_tree_node<T>> right;
    T data;
};

```

//бинарное дерево

```

template<typename T>
class bin_tree{
public:
    std::shared_ptr<bin_tree_node<T>> head;
    //реализация конструктора (валидация строки)
    bin_tree(std::string &str){
        int index = 0;
        bool check = second_checker(str);
        if (check){
            head = create_bin_tree(str, index);
        } else {
            std::cout << "wrong string\n";
            head = nullptr;
        }
    }
    ~bin_tree() = default;
}

```

```

//создание бинарного дерева
std::shared_ptr<bin_tree_node<T>>
create_bin_tree(std::string &str, int &index){
    std::shared_ptr<bin_tree_node<T>> node =
std::make_shared<bin_tree_node<T>>();
    index++;
    if (str[index] == ')'){
        index++;
        return nullptr;
    } else {
        if (int_checker(str)) {
            std::string check_num = "";
            while (str[index] != '(' && str[index]
!= ')') {
                check_num += str[index];
                index++;
            }
            node->data = strtol(check_num.c_str(),
NULL, 10);

        } else if (char_checker(str)){
            node -> data = str[index];
            index++;
        }
    }

    if (str[index] == ')'){
        node -> left = nullptr;
        node -> right = nullptr;
        index++;
        return node;
    } else {

```

```

        node -> left = create_bin_tree(str,
index);
        node -> right = create_bin_tree(str,
index);

        index ++;
        return node;
    }
}

```

//выполнения задания лабораторной (посредством обхода в ширину)

```

void show_tree(){
    std::cout << "success: ";
    std::queue<std::shared_ptr<bin_tree_node<T>>>
queue;

    queue.push(head);
    while (!queue.empty()) {
        std::shared_ptr<bin_tree_node<T>> tmp =
queue.front();

        queue.pop();
        std::cout << tmp->data << " ";
        if (tmp -> left) {
            queue.push(tmp->left);
        }

        if (tmp -> right) {
            queue.push(tmp->right);
        }

    }
}
}

```

```

//LKP обход
void LKP(std::shared_ptr<bin_tree_node<T>> hd){
    if (hd -> left){
        LKP(hd -> left);
    }
    std::cout << hd -> data << " ";
    if (hd -> right){
        LKP(hd -> right);
    }
}

//вставка элемента
void insert_elem(std::shared_ptr<bin_tree_node<T>>
hd, T elem){
    while(hd->left) {
        hd = hd->left;
    }

    hd->left =
std::make_shared<bin_tree_node<T>>();
    hd->left->left = nullptr;
    hd->left->right = nullptr;
    hd->left->data = elem;
}

//удаление элемента
void delete_elem(std::shared_ptr<bin_tree_node<T>>
hd, T elem){
    if(hd->left != nullptr && hd->left -> data ==
elem){
        hd->left = nullptr;
    }
}

```

```

        if(hd -> right != nullptr && hd -> right ->
data == elem){
            hd -> right = nullptr;
        }

        if(hd -> left != nullptr)
            delete_elem(hd -> left, elem);
        if(hd -> right != nullptr)
            delete_elem(hd -> right, elem);
    }

};

```

```

int main(int argc, char* argv[]){
    if(argc == 1){
        std::cout << "Wrong expression\n";
        return 0;
    }
    std::string str(argv[1]);
    if (char_checker(str)){
        bin_tree<char> bt(str);
        if (bt.head){
            bt.show_tree();
        }
    } else if (int_checker(str)){
        bin_tree<int> bt(str);
        if (bt.head){
            bt.show_tree();
        }
    } else {

```

```
        std::cout << "wrong string\n";  
    }  
}
```

ПРИЛОЖЕНИЕ Б

ТЕСТИРОВАНИЕ

Результаты тестирования представлены в таблице Б.1

Таблица Б.1 — Результаты тестирования

№ п/п	Входные данные	Выходные данные	Результат проверки
1.	(a(b)(c))c	success: a b c	success
2.	(v(c(n)(x))())c	success: v c n x	success
3.	(10(100(30(40)(70))))(12))i	success: 10 100 12 30 40 70	success
4.	(a(b)(c)))c	wrong string	wrong string
5.	(1y)(09)9i	wrong string	wrong string
6.	(a(b)(c))i	wrong string	wrong string