

**МИНОБРНАУКИ РОССИИ**  
**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ**  
**ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**  
**«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)**  
**Кафедра математического обеспечения и применения ЭВМ**

**ОТЧЕТ**  
**по практической работе №4**  
**по дисциплине «Алгоритмы и структуры данных»**  
**Тема: Бинарное дерево и лес**

Студентка гр. 7382

\_\_\_\_\_

Еременко А.А

Преподаватель

\_\_\_\_\_

Фирсов М.А.

Санкт-Петербург  
2018

## **Цель работы.**

Ознакомиться с такой структурой данных, как бинарное дерево и методами работы с ними.

## **Постановка задачи.**

Вариант 4-В

Задано бинарное дерево  $b$  типа BT с произвольным типом элементов. В Используя очередь и операции над ней (см. 2), 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.

## **Ход работы.**

### **Описание алгоритма.**

#### **Общий алгоритм:**

Программа считывает входные данные и проверяет их корректность. Если корректны, то идем дальше: каждого брата записываем в индекс родителя умноженного на два плюс два, а каждого сына записываем в индекс родителя умноженного на два плюс 1. Таким образом у нас получается бинарное дерево на основе массива такое, что никогда один элемент не заполнит место другого элемента.

#### **Пояснение к алгоритму:**

Под массив выделяется столько памяти, сколько могло бы потребоваться в худшем случае. Т.к. объем растет со скоростью  $2^n$ , то лучше не использовать данную реализацию для деревьев где очень много братьев. Плюс такой реализации состоит в том что мы можем напрямую обращаться к любому элементу в отличие от реализации бинарного дерева на основе указателей.

### **Описание основных функций алгоритма:**

```
bool Bintree<T>::Isincorrecttext()
```

Данная функция проверяет исходный текст на корректность. Она выдает ошибку когда мы встречаем «)(»или когда закрывающих скобок больше открывающих, а также еще в некоторых случаях, которые описаны в приложении.

```
pushbintree(int index, int index_place, int level)
```

Данная функция заполняет массив элементами из строки так ,чтобы массив можно было представить как бинарное дерево. Принцип куда вставлять элемент описан в «Общий алгоритм».

```
void Bintree<T>::createbintree()
```

Эта функция лишь выделяет достаточное кол-во памяти для массива ,который должен содержать структуру. Худший случай равен  $2^n$ , где n - число всех элементов.

```
void Queue<V>::push(V element)
```

Функция «пушит» элемент в очередь и т. к. не разрешено использовать контейнеры, то пришлось написать массив который динамически сам выделяется по мере нуждаемости.

```
void Queue<V>::fulling_queue(V*text,int len)
```

Данная функция выделяет элементы по уровню из бинарного дерева. Описание как определять откуда брать элементы написана в комментариях к этой функции.

```
V Queue<V>::pop()
```

Функция выдает последний элемент,а также «удаляет» его из очереди. На самом деле функция не удаляет на элемент а лишь ее итератор меняет свое место и поэтому тот символ сохраняется,хотя если после еще запустить элементы, то он будет изменен.

### **Тестирование.**

Были написаны 5 тестов для данной программы, а также 2 скрипта для тестирования и компиляции программы. Тестирование программы для 4 теста представлено на рис. 1.

Исходные данные: (ab(c(de)f))

Результат работы программы:

```
light5551@light5551-ThinkPad-E470: ~/AiSD/7382/EremenkoA/lab4
Test 4:
(ab(c(de)f))
Testing:
Вид бинарного дерева задается вот так-сын обозначается в скобочках, а брат обозна
чается на одном уровне с родителем. пример: (a(b)c) - b сын a, a с брат a
ENTER: (ab(c(de)f))
binary tree:
{a}[0] {}[1] {b}[2] {}[3] {}[4] {c}[5] {}[6] {}[7] {}[8] {}[9] {}[10] {d}[11] {f
}[12] {}[13] {}[14] {}[15] {}[16] {}[17] {}[18] {}[19] {}[20] {}[21] {}[22] {}[2
3] {e}[24] {}[25] {}[26] {}[27] {}[28] {}[29] {}[30] {}[31] {}[32] {}[33] {}[34]
{}[35] {}[36] {}[37] {}[38] {}[39] {}[40] {}[41] {}[42] {}[43] {}[44] {}[45] {}
[46] {}[47] {}[48] {}[49] {}[50] {}[51] {}[52] {}[53] {}[54] {}[55] {}[56] {}[57
] {}[58] {}[59] {}[60] {}[61] {}[62] {}[63] {}[64]
-----
QUEUE:
pushing =a
pushing =b
pushing =c
pushing =d
pushing =f
pushing =e
a
b
c
df
e
```

Рисунок 1 – тестирование

Все тесты представлены в приложении ТЕСТЫ

## Выводы.

В результате работы была изучена структура данных – бинарное дерево на основе массива. А также созданы методы для работы с этой структурой данных. Был использован алгоритм создания бинарного дерева.

## ПРИЛОЖЕНИЕ А

### ИСХОДНЫЙ КОД ПРОГРАММЫ

```
include  
"Bintree.h"
```

```
template <typename T>  
Bintree<T>::Bintree()//начальная инициализация  
{  
    numberofbrackets = 0;  
    len = 0;  
    index_of_str = 0;  
}
```

```
template <typename T>  
Bintree<T>::~~Bintree()  
{  
    delete[] bintree;//удаление дерева  
}
```

```
template <typename T>  
void Bintree<T>::settext()  
{  
    std::getline(std::cin, text);  
}
```

```
template<typename T>  
void Bintree<T>::gettext()  
{  
    std::cout << text << "\n";  
}
```

```
template<typename T>  
bool Bintree<T>::Isincorrecttext()  
{  
    int brackets = 0;  
    if (text[0] != '(') {  
        return false;  
    }  
    for (int i = 0; i < text.size(); i++)  
    {  
        if (brackets < 0)//на случай )(  
        {  
            std::cout << "NOT correct bintree\n";  
            return false;  
        }  
        if (text[i] == '(')  
        {  
            numberofbrackets++;  
            if (i)  
            {  
                if (text[i - 1] == ')')//на случай )(  
                {
```

```

if (brackets)
{
return false;
}
}
}
brackets++;
continue;
}
if (text[i] == ')')
{
numberofbrackets++;
brackets--;
continue;
}
}
if (brackets)
return false;
return true;
}

template<typename T>
void Bintree<T>::pushbintree(int index, int index_place, int level) //start 0 0 1
{
//int level_brackets = level - 1;
int prev = index_place;
bool first_bracket = true;
while (index_of_str < text.length())
{
if (text[index_of_str] == '(')
{
if (!first_bracket)//1 скобку на считаем за сына
{ //prev ýòî ðàññîëîæååîèäååûåóååî ýèåîå
pushbintree(index, prev * 2 + 1, level + 1); //каждый сын находится на индексе
родителя умноженного на 2 + 1
continue;
}
else
{
index_of_str++;
first_bracket = false;
continue;
}
}
if (text[index_of_str] == ')')//при встрече ')'выходим из рекурсии
{
index_of_str++;
return;
}
prev = index_place;

```

```

add(index_place, text[index_of_str]); index_of_str+
+;
continue;
}
}
template<typename T>
void Bintree<T>::add(int & index_place, T element)
{
bintree[index_place] = element;
index_place = index_place * 2 + 2; //каждый брат будет находиться на
индексе родителя умноженного на 2 + 2
}
template<typename T>
void Bintree<T>::createbintree()
{
bintree = new T[(int)pow(2, text.length() - numberofbrackets) + 1](); //идет расчет
на самый худший случай
}
template<typename T>
void Bintree<T>::getbintree()
{
len = pow(2, text.length() - numberofbrackets) + 1;
for (int i = 0; i < len; i++)
std::cout << "{" << bintree[i] << "}" << "[" << i << "]" ";
std::cout << "\n";
}

```

@@ -0,0 +1,26 @@

```

#pragma once #include <string> #include
<iostream> #include <math.h>
template <typename T> class Bintree
{
public: Bintree();
~Bintree(); void settext(); void gettext();
bool Iscorrecttext(); //проверка на корректность
void pushbintree(int index, int index_place, int level);
void createbintree();
void getbintree(); int len;
T* bintree; //бинарное дерево на основе массива
private:

```

```

int numberofbrackets;//кол-во скобок std::string
text;//исходный текст из cout int index_of_str;
void add(int&index_place,T element);
};

```

```

@@ -0,0 +1,65 @@
#include "Queue.h"
template<typename V>
Queue<V>::Queue()//создаем начальную очередь и переменные
{
index = 0;
size = 10;
lenght = 0;
queue = new V[size];
}
template<typename V>
void Queue<V>::push(V element)
{
if (lenght == size)//логика такая же как у реаллока в си
{
// realloc
V* time_queue = new V[size]; std::memcpy(time_queue, queue, (size) * sizeof(V));
delete[] queue;
queue = new V[size + 10];

std::memcpy(queue, time_queue, (size) * sizeof(V)); delete[] time_queue;
size += 10;
}
queue[lenght++] = element;
}

template<typename V> V Queue<V>::pop()
{
return queue[index++];
}

template<typename V>
void Queue<V>::fulling_queue(V*text,int len)
{
int j = 1;
int block = 1;
for (int i = 0;i < len;i++)
{
if (i==block)//каждый уровень дерева находится в промежутке [2^(n-1);2^n]
{
push('\n');
}
}
}

```



```

block += pow(2, j++);
}

if(!isalpha(text[i]))//пушим только существующий символ continue;
std::cout << "pushing =" << text[i] << "\n"; push(text[i]);
}
}

template<typename V> void Queue<V>::getqueue()
{
while (index!=length)//индекс изменяется в функции pop std::cout << pop();
}

template<typename V> Queue<V>::~~Queue()
{
delete[]queue;//удаляем очередь
}

```

@@ -0,0 +1,21 @@

```

#include <algorithm> #include <iostream> #include
<cstring> template<typename V> class Queue
{
public: Queue();
~Queue();

void push(V element); V pop();
void fulling_queue(V*text,int len);//заполнение очереди
void getqueue();//вывод очереди
private: int index; V*queue; int size, length;
};

```

```

#include <iostream> #include "Bintree.cpp"

```

## ТЕСТЫ

Test 1:

(a(b(c)))

---

Testing:

Вид бинарного дерева задается вот  
так-сын обозначается в скобках, а  
брат обозначается на одном уровне  
с родителем. пример: (a(b)c) - b сын a, a  
с брат a

ENTER:(a(b(c)))

binary tree:

---

QUEUE:

pushing =a

pushing =b

pushing =c

a

b

c

---

Test 2:

---

Testing:

Вид бинарного дерева задается вот  
так-сын обозначается в скобках,а  
брат обозначается на одном уровне  
с родителем.пример:(a(b)c)-b сын a,a  
с брат a

ENTER:

wrong

---

Test 3:  
(saa(ds)(ds))

---

Testing:

Вид бинарного дерева задается вот  
так-сын обозначается в скобках, а  
брат обозначается на одном уровне  
с родителем. пример: (a(b)c) - b сын a, a  
с брат a

ENTER: (saa(ds)(ds))  
wrong

---

Test 4:  
(ab(c(de)f))

---

Testing:

Вид бинарного дерева задается вот  
так-сын обозначается в скобочках, а  
брат обозначается на одном уровне  
с родителем. пример: (a(b)c) - b сын a, a  
с брат a

ENTER: (ab(c(de)f))  
binary tree:

---

QUEUE:

pushing =a  
pushing =b  
pushing =c  
pushing =d  
pushing =f  
pushing =e  
a  
b  
c  
df  
e

---

Test 5:  
(a(bc(dr)e)gh)

---

Testing:

Вид бинарного дерева задается вот  
так-сын обозначается в скобочках, а  
брат обозначается на одном уровне  
с родителем. пример: (a(b)c) - b сын a, a  
с брат a

ENTER: (a(bc(dr)e)gh)

binary tree:

---

QUEUE:

pushing =a

pushing =b

pushing =g

pushing =c

pushing =h

pushing =d

pushing =e

pushing =r

a

bg

ch

de

r