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

ОТЧЕТ

по лабораторной работе №4

по дисциплине «Алгоритмы и структуры данных»

Тема: Деревья

Студентка гр. 7383 _____ Прокопенко Н.

Преподаватель _____ Размочаева Н. В.

Санкт-Петербург

2018

Содержание

Цель работы

Познакомиться со структурами данных и научиться реализовывать их на языке программирования C++.

Формулировка задачи: задано бинарное дерево b типа BT с произвольным типом элементов. Используя очередь и операции над ней, 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.

Реализация задачи

Входные данные: бинарное дерево *b* типа *BT* с произвольным типом элементов, бинарное дерево задается из файла.

В данной работе было написано несколько функций для реализации задачи. Перечень функций:

int main ( ) - головная функция. В данной функции выводится меню программы, после чего считывается выбранный вариант и запускается функция switch( ), которая находясь в теле цикла while( ), будет выполняться до тех пор, пока пользователь не захочет выйти из программы.

При введении пользователем «1», программа предлагает ввести имя входного файла, и если файл с таким файлом существует, то вызывается функция enterBT(int index, Tree<T>** b, istream &input) которая обрабатывает строку из файла и формирует дерево. Далее вызывается функция основная функция goriz(int index, Tree<T>** b, ofstream &fout), которая выполняет алгоритм обхода в ширину.

При введении пользователем «2», происходит выход из программы.

При введении пользователем других данных предлагается выбрать один из пунктов меню, описанных выше.

void goriz(int index, Tree<T>** b, ofstream &fout) – основная функция, которая выполняет алгоритм обхода в ширину. В функцию goriz передаем индекс дерева. В очередь передаем этот индекс, пока очередь не пуста

будем циклически выполнять следующий алгоритм:

1. Берем из очереди индекс узла дерева,
2. С помощью этого индекса выводим на экран или в файл корень,
3. Если левое поддерево существует, то заносим его индекс в очередь,
4. Если правое поддерево существует, то заносим его индекс в очередь.

Методы класса Queue, для работы с очередью:

~Queue(); //деструктор

void Put(T x);//добавление нового элемента в конец очереди

T Get();//возвращение значения первого элемента и его удаление

int Kol();//возвращает количество элементов в очереди

bool Empty();//возвращает True если очередь пуста.

Методы класса Tree, для формирования и работы с деревом:

bool isNull()//возвращает flag

void SetInfo(T i)//запись индекса корня поддерева

void SetLeft(int l) //запись индекса левого узла

void SetRight(int r) //запись индекса правого узла

T GetInfo() //взятие корня поддерева

int Left() //взятие индекса левого узла

int Right() //взятие индекса правого узла

Код программы представлен в Приложении А.

Тестирование

Программа собрана в операционной системе Ubuntu 17.04 с использованием компилятора g++. В других ОС и компиляторах тестирование не проводилось. Результаты тестирования показали, что поставленная цель выполнена. Результаты тестирования представлены в Приложении Б.

Выводы

В ходе выполнения лабораторной работы были изучены основные понятия стеке, был реализован стек на базе массива на языке

программирования C++. Также была написана программа для записи выражения из постфиксного в инфиксный вид.

ПРИЛОЖЕНИЕ А. Код программы

main.cpp

```
/*Скобочное представление бинарного дерева (БД):
< БД > ::= < пусто > | < непустое БД >,
< пусто > ::= /,
< непустое БД > ::= ( < корень > < БД > < БД > )*/

#define _CRT_SECURE_NO_WARNINGS
#include <iostream>
#include <stdlib.h>
#include <fstream>
#include <cstdlib>
#include "btree.h"
#include "queue.h"
using namespace std;
template <typename T>
void goriz(int index, Tree<T>** b, ofstream &fout)
{
    Queue <int> q;
    q.Put(index);//заношу в очередь индекс корневого элемента дерева
    while (!q.Empty())// пока очередь не пуста
    {
        index = q.Get();//Убираем из очереди индекс элемента дерева
        fout << b[index]->GetInfo() << " ";
        if (!b[index]->isNull()) //если слева есть элемент то заносим
его индекс в очередь
        {
            q.Put(b[index]->Left());
        }
        if (!b[index]->isNull())//если справа есть элемент то заносим
его индекс в очередь
        {
            q.Put(b[index]->Right());
        }
    }
}


int main() {
    Tree<int> *b[100];
    for(int i=0; i<100;i++){
        b[i]= new Tree<int>();
    }
    int input;
    string file_name;
    fstream file;
    ofstream fout;
    while(true){
        cout << "------------------------------\n";
        cout << "Входные данные \n";
        cout << "1: из файла \n2: выход \n";
```

```
        cin >> input;
        switch (input)
        {
        case 1:
            cout << "Enter the name of the file:" << '\n';
            cin >> file_name;
            file.open(file_name, fstream::in);
            fout.open("output.txt");
            if (!file.is_open() || !fout.is_open()) {
                cout << "Error opening file.\n";
                break;
            }
            else{
                enterBT(0, b, file);
                goriz(0, b, fout);
                cout << endl << "вывод ответа в файле\n";
            }
            file.close();
            fout.close();
            break;

        case 2:
            for(int i=0; i<100;i++){
                delete b[i];
            }
            return 0;

        default:
            cout << "Enter again.\n";
            break;
        }
    }
return 0;
}
```

queue.h

```
#ifndef QUEUE_H
#define QUEUE_H
#include <iostream>
#include <cstdlib>
using namespace std;
template <typename T>
struct Nod            //Структура для очереди
{
    T znach;
    Nod *Next;
};

template <typename T>
class Queue           //Очередь
{
    Nod<T> *First, *Last;
```

```
    int kol;
public:
    Queue() :First(NULL), Last(NULL) { kol = 0; } //конструкор
    ~Queue(); //деструктор
    void Put(T x);//добавление нового элемента в конец очереди
    T Get();//возвращение значения первого элемента и его удаление
    int Kol();//возвращает количество элементов в очереди
    bool Empty();//возвращает True если очередь пуста
};
template <typename T>
void Queue<T>::Put(T x) //добавление нового элемента в конец очереди
{
    Nod<T> *temp = new Nod<T>;//создание указателя на структуру Nod и
выделение под него памяти
    temp->znach = x;
    temp->Next = NULL;
    if (First == NULL)
    {
        First = temp;
        Last = temp;
        kol++;
    }
    else
    {
        Last->Next = temp;
        Last = temp;
        kol++;
    }
};

template <typename T>
Queue<T>::~Queue() //удаление всех эелементов
{
    Nod<T> *temp;
    kol = 0;
    while (First != NULL)
    {
        Nod<T> *temp;
        temp = First;
        First = First->Next; //переходим на следующий элемент
        delete temp;
    }
};

template <typename T>
T Queue<T>::Get() //возвращение значения первого элемента и его удаление
{
    if (First == NULL) { printf("Очередь пуста \n"); exit(1); }
    Nod<T> *temp;
    T x;
    x = First->znach;
    temp = First;
```

```
    First = First->Next;
    delete temp;
    kol--;
    return x;
};

template <typename T>
int Queue<T>::Kol() //возвращает количество элементов в очереди
{
    return kol;
};

template <typename T>
bool Queue<T>::Empty() //возвращает True если очередь пуста
{
    return First == NULL;
};

#endif // QUEUE_H
```

btree.h

```
#ifndef BTREE_H
#define BTREE_H
#include <iostream>
#include <fstream>
using namespace std;


template <typename T>
class Tree      //Структура дерева
{
private:
    T info = 0;
    int lt; //индекс левого узла
    int rt; //индекс правого узла
public:
    bool flag = false;

    bool isNull(){
        return flag;
    }
    void SetInfo(T i){
        info = i;
    }
    void SetLeft(int l){
        lt = l;
    }
    void SetRight(int r){
        rt = r;
    }
    T GetInfo() //взятие корня поддерева
    {
```

```
        return info;
    }

    int Left() //взятие индекса левого узла
    {
        return lt;
    }

    int Right() //взятие индекса правого узла
    {
                return rt;
    }



};

template <typename T>
int enterBT(int index, Tree<T>** b, istream &input) // считывание данных
из файла
{
    int cur = index;
    T ch;
    input >> ch;
    b[index]->SetInfo(ch);
    if (ch == 0)
    {
        b[cur]->flag = true;
        return index;
    }
    else
    {
        index++;
        (b[cur])->SetLeft(index);
        index = enterBT(index, b, input);
        index++;
        (b[cur])->SetRight(index);
        index = enterBT(index, b, input);
        return index;
    }
}

int enterBT(int index, Tree<char>** b, istream &input) // считывание
данных из файла
{
    int cur = index;
    char ch;
    ch = input.get();
    (b[cur])->SetInfo(ch);
    if (ch == '/')
    {
        b[cur]->flag = true;
```

```
        return index;
    }
    else
    {
        index++;
        (b[cur])->SetLeft(index);
        index = enterBT(index, b, input);
        index++;
        (b[cur])->SetRight(index);
        index = enterBT(index, b, input);
        return index;
    }
}

#endif // BTREE_H
```

ПРИЛОЖЕНИЕ Б. Тестовые случаи

Таблица 1 - Результаты тестов.

Input	Output	True/False
fghj///k///	fghkj	True
abgm///sd//r//hf//k//	abhgsfkmdr	True
rty//i////	rtyi	True
oiuy//i///k////	oikuyi	True
f//	f	True