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 _____ Боблаков Д.С.

Преподаватель _____ Филатов А.Ю.

Санкт-Петербург

2020

Цель работы

Изучить такую структуру данных, как бинарное дерево. Получить навыки для работы с бинарными деревьями. Реализовать бинарное дерево на языке программирования C++.

Задание

Вариант 10y.

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

- восстановить дерево b и вывести его изображение;
- перечислить узлы дерева b в порядке КЛП.

Описание алгоритма работы

Сначала программа считывает две строчки: записи дерева в порядке ЛКП и ЛПК. Далее эти две строчки проверяются на корректность с помощью функции isCorrect. В случае некорректного ввода данных программа выведет сообщение об ошибке и завершит свою работу. В ином случае программа с помощью функции createTree() рекурсивно строит бинарное дерево MyBinTree по следующему алгоритму: после каждого вызова данной функции в последовательности ЛПК удаляется последний символ, а в последовательности ЛКП удаляется символ соответствующий удаленному так, чтобы в результате получились две последовательности ЛКП. ЛПК разделяется по аналогичному алгоритму, а получившиеся после очередной итерации строки служат аргументами к следующему вызову этой функции. Затем с помощью рекурсивного обхода выводится запись дерева в порядке КЛП с помощью функции printKLP(). После этого вызывается рекурсивная

функция printTree(), которая печатает дерево в уступчатой форме.

Разработанный код см. в приложении А.

Формат входных и выходных данных

Программа принимает на вход две строки, содержащие последовательности символов – значений узлов дерева в порядке ЛКП и ЛПК: <symbols...>

<symbols...>

Пример:

BAC

BCA

На выходе программа выведет последовательность символов (узлоы дерева) в порядке КЛП, а также изображение дерева в виде уступчатого списка. В случае некорректно введенных данных программа выведет следующее сообщение об ошибке: «Error: incorrect write of the tree!»

Описание основных структур данных и функций

Class Node

Экземпляр данного класса будет являться узлом бинарного дерева. Поля *left* и *right* хранят указатели на левое и правое поддерево соответственно. Поле value хранит значение узла дерева.

Bool IsCorrect()

Данная функция принимает строки lkp и lpk и проверяет их на корректность записи бинарного дерева.

Void createTree()

Данная функция рекурсивно создает дерево, указатель на которое сохраняется в std::shared_ptr<Node> &BinTree.

Void printKLP()

Данный метод печатает запись бинарного дерева в порядке КЛП.

Bool isNull()

Данный метод возвращает истину, если указатель на узел не равен nullptr, в ином случае возвращает ложь.

NodePtr consBT()

Данный метод возвращает указатель на «склеенное» дерево.

Void printTree()

Данный метод печатает бинарное дерево в формате уступчатого списка.

NodePtr copyTree()

Данный метод возвращает указатель на новое дерево, которое является копией другого дерева.

Тестирование

Тестирование программы проводится с помощью bash-скрипта tests_script. Для запуска тестирования необходимо выполнить команду ./tests_scripts, предварительно собрав программу с помощью команды make.
Для каждого теста выводится сообщение Test# <входные данные> - passed или Test# <входные данные> - failed. Также в каждом тесте выводится ожидаемый и полученный результат. Файлы с выходными данными после завершения тестирования удаляются.

Результаты тестирования см. в приложении Б.

Выводы

Была изучена такая структура данных, как бинарное дерево. Были приобретены навыки работы с бинарными деревьями. Был реализован алгоритм построения и обработки бинарного дерева на языке C++.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Файл: main.cpp

```
#include <iostream>
#include <string>
#include "Tree.h"


void createTree( std::string lkp, std::string lpk, char value,
std::shared_ptr<Node> &BinTree)
{
    using NodePtr = std::shared_ptr<Node>;
    NodePtr leftBranch=std::make_shared<Node>();
    NodePtr rightBranch=std::make_shared<Node>();
    leftBranch= nullptr;
    rightBranch= nullptr;

    if (lpk.length() < 2){
        BinTree = std::make_shared<Node>();
        BinTree->value = value;
        return;
    }

    int index = lkp.find(value);
    std::string leftlkp = lkp.substr(0, index);
    std::string leftlpk = lpk.substr(0, index);
    std::string rightlkp = lkp.substr(index + 1);
    lpk = lpk.erase(lpk.length()-1, 1);
    std::string rightlpk = lpk.substr(index);
    if (index!= 0){
        createTree(leftlkp, leftlpk, leftlpk[leftlpk.size() - 1],
leftBranch);
    }
```

```
        createTree(rightlkp, rightlpk, rightlpk[rightlpk.size() - 1],
rightBranch);
        BinTree=BinTree->consBT(value, leftBranch, rightBranch);
    }
    bool isCorrect(std::string lkp , std::string lpk ){
        if (lkp.length() != lpk.length())
            return false;
        int i = 0;
        int pos=0;
        while(lkp.length()>0){
            pos=lpk.find(lkp[0]);
            if (pos==-1){
                return false;
            }
            lpk.erase(pos,1);
            lkp.erase(i,1);
        }
        if (lpk.length() != 0 && lkp.length()!=0){
            return false;
        }
        return true;
    }

    int main(){


        std::string lkp;
        std::string lpk;
        std::cout << "Enter the nodes in the LEFT ROOT RIGHT order\
n";
        std::getline(std::cin,lkp);
        std::cout << "Enter the nodes in the LEFT RIGHT ROOT order\
n";
        std::getline(std::cin,lpk);
        if (!isCorrect(lkp, lpk)){
```

```
            std::cout<<"\nError: incorrect write of the tree!\n";
            return EXIT_FAILURE;
        }
        using NodePtr = std::shared_ptr<Node>;
        NodePtr MyBinTree=std::make_shared<Node>();
        createTree(lkp, lpk, lpk[lpk.size() - 1], MyBinTree);
        std::cout << " the nodes in the ROOT LEFT RIGHT order:" << "\
n";
        MyBinTree->printKLP(MyBinTree);
        std::cout <<"\n";
        std::cout << "Picture of a Binary Tree:" << "\n";
        MyBinTree->printTree(MyBinTree,1);
        return 0;
    }
```

Файл: Tree.cpp

```
    #include "Tree.h"
    using NodePtr = std::shared_ptr<Node>;
    bool Node::isNull(std::shared_ptr<Node> node){
        return (node == nullptr);
    }

    NodePtr Node::consBT(const char& x, std::shared_ptr<Node>
leftBranch, std::shared_ptr<Node> rightBranch){
        NodePtr unitedNode=std::make_shared<Node>();
        unitedNode->value = x;
        unitedNode->left = std::move(leftBranch);
        unitedNode->right = std::move(rightBranch);
        return unitedNode;
    }

    void Node::printTree(std::shared_ptr<Node> node, int n){
        if (node!= nullptr) {
            std::cout << ' ' << node->value;
```

```
        if(!isNull(node->right)) {
            printTree (node->right,n+1);
        }
        else std::cout <<"\n";

        if(!isNull(node->left)) {
            for (int i=1;i<=n;i++){
                std::cout << "  ";
            }
            printTree (node->left,n+1);
        }
    }
    else {
        std::cout<<"Tree is empty!";
    }
}

NodePtr Node::copyTree(const NodePtr &head) {
    NodePtr NewHead = std::make_shared<Node>();
    if (head== nullptr){
        return nullptr;
    }
    NewHead->value=head->value;
    NewHead->left=copyTree(head->left);
    NewHead->right=copyTree(head->right);

    return NewHead;
}
NodePtr Node::moveTree(NodePtr &head) {
    NodePtr NewHead = std::make_shared<Node>();
    if (head== nullptr){
        return nullptr;
    }
    NewHead=head;
    head= nullptr;
```

```
    return NewHead;
}

Node::Node() {
    this->value='\0';
    this->right = nullptr;
    this->left = nullptr;
}

void Node::printKLP(NodePtr head) {
    if (head!= nullptr){
        std::cout<<head->value;
    }
    if(head->left!= nullptr){
        printKLP(head->left);
    }
    if (head->right!= nullptr){
        printKLP(head->right);
    }
}
//
//Node &Node::operator=(const NodePtr &other) {
//   *this=other->copyTree(other);
//    return *this;
//}
//
//Node::Node(std::shared_ptr<Node> head) {
//   *this=this->copyTree((const NodePtr &) head);
//}


//Node& Node::operator=(NodePtr &&other) {
//    this=std::move(other);
//    return this;
//}
```

Файл: Tree.h

```
#pragma once
#include <iostream>
#include <memory>

class Node {

public:
    using NodePtr = std::shared_ptr<Node>;
    char value;
    NodePtr left;
    NodePtr right;
    Node();
    void printKLP(NodePtr head);
    bool isNull(NodePtr node);
    NodePtr consBT(const char& x, NodePtr, NodePtr);
    void printTree(std::shared_ptr<Node> node, int n);
    NodePtr copyTree(const NodePtr& head);
   // Node& operator=(const NodePtr& other);
    NodePtr moveTree(NodePtr &head);
    //Node(std::shared_ptr<Node> head);
};
```

Файл: Makefile

```
  lab3: Source/main.cpp
        g++ -std=c++17 Source/main.cpp Source/Tree.cpp -o lab3

run_tests: lab3
        ./tests_script
```

Файл: tests_scripts.sh

```
#! /bin/bash

printf "\nRunning tests...\n\n"
for n in {1..10}
```

```
    do
        ./lab3 < "./Tests/tests/test$n.txt" > "./Tests/out/out$n.txt"
        printf "Test$n:\n"
        cat "./Tests/tests/test$n.txt" #| tr -d '\n'
        if cmp "./Tests/out/out$n.txt"
"./Tests/true_results/true_out$n.txt" > /dev/null; then
                printf "*** - Passed\n"
        else
                printf "*** - Failed\n"
        fi
        printf "Desired result:\n"
        cat "./Tests/true_results/true_out$n.txt"
        printf "Actual result:\n"
        cat "./Tests/out/out$n.txt"
        printf "==============\n"
    done
    rm ./Tests/out/out*
```

ПРИЛОЖЕНИЕ Б

ТЕСТИРОВАНИЕ

Результаты тестирования представлены в таблице Б.1

Таблица Б.1 — Результаты тестирования

№ п/п	Входные данные	Выходные данные	Комментарии
1.	BCA BAC	CBA C A B	Простейшее бинарное дерево
2.	908 980	098 0 8 9	Простейшее бинарное дерево
3.	dbeafcg debfgca	abdecfg a c g f b e d	Бинарное дерево глубиной 3
4.	251637 526731	125367 1 3 7 6 2 5	Неполное бинарное дерево глубиной 3, в котором отсутствует элемент у левого-левого узла.
5.	13789 98731	13789 1 3 7 8 9	Бинарное дерево, которое имеет только праве поддеревья.
6.	Trump will win! Biden will lose!	Error: incorrect write of the tree!	Входные строки, не являющиеся записями дерева.
7.	a+b*c-d/e+f*g ab+c*defg*+/-	-*+abc/d+e*fg - / + * g f	Представление математического выражения бинарным

		e d * c + b a	деревом.
8.	123456 654322	Error: incorrect write of the tree!	Получены противоречивые входные данные.
9.	qwertyuiop poiuytrewq	qwertyuiop q w e r t y u i o p	Бинарное дерево, только с правыми поддеревьями.
10.	qweasdzxc qwezxcasde	Error: incorrect write of the tree!	На вход получены две строки разной длины.