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

ОТЧЕТ

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

Студент гр. 9304 _____ Афанасьев А.

Преподаватель _____ Филатов А.Ю.

Санкт-Петербург
2020

Цель работы.

Ознакомиться с понятием бинарного дерева, реализовать его, решить поставленную задачу на его основе.

Постановка задачи.

Вариант *12y*.

Формулу вида:

<формула> ::= <терминал> | (<знак> <формула> <формула>)

*<знак> ::= + | – | **

<терминал> ::= 0 | 1 | ... | 9 | a | b | ... | z

можно представить в виде бинарного дерева.

– формула из одного терминала представляется деревом из одной вершины с этим терминалом;

– формула вида (s f_1 f_2) представляется деревом, в котором корень – это знак *s*, а левое и правое поддеревья – соответствующие представления формул f_1 и f_2.

Требуется:

– построить дерево *t* из строки, задающей формулу в префиксной форме (перечисление узлов *t* в порядке КЛП);

– преобразовать дерево-формулу *t*, заменяя в нем все поддеревья, соответствующие формуле *(f + f)*, на поддеревья, соответствующие формуле *(2 * f)*.

Выполнение работы.

Программа на вход ожидает строку для преобразованию. На рисунке 1 представлен пример запуска программы.

```
./lab3 "++dab"
```

Рисунок 1 - Запуск программы

Класс *BinarySearchTreeNode.*

В нем хранится шаблонное поле *obj*, *weak_ptr* на родителя *parent* и *shared_ptr* на левого и правого потомков (*left* и *right*). Сам класс является шаблонным.

Класс *BinarySearchTree.*

Шаблонный класс бинарного дерева поиска. Имеет поле *shared_ptr* на голову дерева *head*. Имеет конструктор от строки-формулы и метод упрощения в соответствии задаче, доступные только при типе *char*. Также есть перегруженные операторы копирования, перемещения, конструкторы копирования, перемещения для любого подставляемого типа. Реализованы методы префиксного, инфиксного и постфиксного обходов, метод проверки на пустоту *empty()*, метод вставки элемента *insert()* и удаления элемента *delete()* и оператор вывода в поток с помощью префиксного обхода. Также были реализованы вспомогательные приватные методы, о некоторых из которых позже.

Метод *simplifyTheFormula()* вызывает рекурсивный вспомогательный приватный метод *recursiveSimplifyTheFormula()*, который принимает аргументом *shared_ptr* на текущий элемент, который будет проверяться на возможность упрощения, если такая возможность есть, то он поменяет операцию в текущем элементе на умножение и у левой формулы этого умножения удалит всех потомков и и вставит в нее 2. Потом в любом случае он вызовет себя от всех потомков текущего элемента, чтобы упростить оставшиеся подходящие элементы. Рекурсия прерывается, если *shared_ptr* на текущий элемент равен *nullptr*.

Конструктор от строки ждет аргументом строку-формулу. Он вызывает приватный метод *recursiveConstructorFromStr()*, который принимает строку-формулу, указатель на текущий элемент *ptrNode*, куда нужно будет вписать значение из строки и указатель на его родителя, потому что элемент в *ptrNode* может еще не существовать. Если во время чтения строки встречается знак, то

функция определит текущий элемент этим знаком, и вызовет саму себя от левого потомка, а потом от правого потомка, а если знак не встретился, то она просто записывает этот символ в текущий элемент и завершается. Проверкой полученного дерева на корректность занимается приватный метод *checkTaskBin()*, который проверяет, что у каждого терминала нет потомков, а у каждой операции два потомка. Если он возвращает логический нуль, то дерево остается пустым. Если же дерево собралось корректно, но строка была неправильной то дерево тоже останется пустым.

Класс *MyException* был создан для обработки синтаксических ошибок ввода и для обработки использования методов при неподдерживаемом подставленном типом, он в зависимости от типа ошибки выдает пользователюсоответствующее сообщение о ней. Сами исключения бросаются в процессевычисления логического выражение и конструирования объекта.

Исходный код находится в приложении А.

Тестирование.

Программу можно собрать командой *make*, после этого создается исполняемый файл *lab3*. Его можно запустить, передав в него строку-формулу. Также можно запустить тестирующий скрипт *testScript.py,* конфигурационный файл которого лежит в папке с исполняемым файлом. В конфигурационном файле можно настроить многие параметры, включая количество тестов и директорию, в которой они находятся. В тестовом файле должна находиться только лишь одна строка – сам тест. Программа возвращает сообщение об синтаксической ошибке ввода,если такая была, либо ответ. Тестирующий скрипт выводит на экран поданную строку, результат работы программы, правильный ответ и *success* или *fail* в зависимости от совпадения того, что вернула программа, и правильного ответа. Пример его работы можно посмотреть на рисунке 2. А в таблице 1 можно посмотреть примеры строк-тестов.

```
strx@strxpc:~/gitreps/main/Programs/ETU/3SEM/AaDS/1b3$ python testScript.py
Make sure that this script is in the same directory as the program execute file.

test0:
Input: "-*+ab+aa/d+e*+aag"
CorrectAnswer: - * * 2 b * 2 a / d * 2 * * 2 a g
Answer: - * * 2 b * 2 a / d * 2 * * 2 a g
Result: success

test1:
Input: "+ab"
CorrectAnswer: * 2 b
Answer: * 2 b
Result: success

Total: Successes: 2. Fails: 0
```

Рисунок 2 - Пример работы скрипта

Таблица 1. Примеры входных и выходных данных

№ теста	**Входные данные**	**Выходные данные**
0	"-*+ab+aa/d+e*+aag"	- * * 2 b * 2 a / d * 2 * * 2 a g
1	"+ab"	* 2 b
2	"a"	a
3	"+++adfg"	* 2 g
4	"-+i3/-14+*af5"	- * 2 3 / - 1 4 * 2 5
5	"+*-+23b1/f-r3"	* 2 / f - r 3
6	"+*-+23b1/f-r33"	Wrong string
7	"+"	Wrong string
8	"2d"	Wrong string
9	""	

Выводы.

В ходе выполнения лабораторной работы было реализовано шаблонное бинарное дерево поиска и решена поставленная задача на его основе. Была разработана программа, упрощающая выражение, введенное в КЛП виде.

Использование бинарного дерева поиска, основанного на указателях, оправдано более удобной работой с элементами, а также малым потреблением памяти относительно реализации на основе массива.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД

main.cpp

```
#include "../libs/BinarySearchTree.h"

#include <iostream>

int main(int argc, char const *argv[])
{
    if (argc >= 2)
    {
        BinarySearchTree<char> treeChar(argv[1]);
        treeChar.simplifyTheFormula();
        std::cout << treeChar;
    }
    return 0;
}
```

BinarySearchTreeNode.h

```
#ifndef __BINARYSEARCHTREENODE__H__
#define __BINARYSEARCHTREENODE__H__

#include <memory>

template <typename T>
class BinarySearchTree;

template <typename T>
class BinarySearchTreeNode
{
    friend class BinarySearchTree<T>;
    T obj;

public:
    std::shared_ptr<BinarySearchTreeNode<T>> left;
    std::shared_ptr<BinarySearchTreeNode<T>> right;
    std::weak_ptr<BinarySearchTreeNode<T>> parent;

    BinarySearchTreeNode(const T &val, const
std::shared_ptr<BinarySearchTreeNode<T>> &ptrParent =
nullptr, const std::shared_ptr<BinarySearchTreeNode<T>>
&ptrLeft = nullptr, const
std::shared_ptr<BinarySearchTreeNode<T>> &ptrRight =
nullptr);
};

template <typename T>
BinarySearchTreeNode<T>::BinarySearchTreeNode(const T &val,
const std::shared_ptr<BinarySearchTreeNode<T>> &ptrParent,
```

```
const std::shared_ptr<BinarySearchTreeNode<T>> &ptrLeft,
const std::shared_ptr<BinarySearchTreeNode<T>> &ptrRight) :
obj(val), left(ptrLeft), right(ptrRight), parent(ptrParent)
{}

#endif //!__BINARYSEARCHTREENODE__H__
```

BinarySearchTree.h

```
#ifndef __BINARYSEARCHTREE__H__
#define __BINARYSEARCHTREE__H__

#include <iostream>
#include <ostream>
#include <string>
#include <algorithm>
#include <vector>

#include "BinarySearchTreeNode.h"
#include "MyException.h"

template <typename T>
class BinarySearchTree
{
    std::shared_ptr<BinarySearchTreeNode<T>> head;

    std::shared_ptr<BinarySearchTreeNode<T>> min(const
std::shared_ptr<BinarySearchTreeNode<T>> &ptrNode);
    void recursivePrefixTraverse(std::vector<T>
&vectorOfNodes, const
std::shared_ptr<BinarySearchTreeNode<T>> &nodePtr) const;
    void recursivePostfixTraverse(std::vector<T>
&vectorOfNodes, const
std::shared_ptr<BinarySearchTreeNode<T>> &nodePtr) const;
    void recursiveInfixTraverse(std::vector<T>
&vectorOfNodes, const
std::shared_ptr<BinarySearchTreeNode<T>> &nodePtr) const;
    void recursiveInsert(const T &val,
std::shared_ptr<BinarySearchTreeNode<T>> &ptrNode, const
std::shared_ptr<BinarySearchTreeNode<T>> &ptrParent =
nullptr);
    void recursiveRemove(const T &val,
std::shared_ptr<BinarySearchTreeNode<T>> &ptrNode, const
std::shared_ptr<BinarySearchTreeNode<T>> &ptrParent =
nullptr);
    std::shared_ptr<BinarySearchTreeNode<T>>
recursiveCopy(const std::shared_ptr<BinarySearchTreeNode<T>>
&ptrNodeToCopy, const
std::shared_ptr<BinarySearchTreeNode<T>> &ptrParent =
nullptr);
    //для задания:
    bool checkTaskBin(const
std::shared_ptr<BinarySearchTreeNode<T>> &ptrNode);
```

```
    std::shared_ptr<BinarySearchTreeNode<T>>
recursiveConstructorFromStr(const std::string &str,
std::shared_ptr<BinarySearchTreeNode<T>> &ptrNode, const
std::shared_ptr<BinarySearchTreeNode<T>> &ptrParent =
nullptr);
    void
recursiveSimplifyTheFormula(std::shared_ptr<BinarySearchTreeN
ode<T>> &ptrNode);
    void deleteSpaces(std::string &str);

    size_t curStrIndex = 0;

public:
    ~BinarySearchTree() = default;
    BinarySearchTree();
    BinarySearchTree(BinarySearchTree<T> &&tree);
    BinarySearchTree(const BinarySearchTree<T> &tree);
    BinarySearchTree<T> &operator=(BinarySearchTree<T>
&&tree);
    BinarySearchTree<T> &operator=(const BinarySearchTree<T>
&tree);
    std::vector<T> prefixTraverse() const;
    std::vector<T> postfixTraverse() const;
    std::vector<T> infixTraverse() const;
    bool empty() const;
    void insert(const T &val);
    void remove(const T &val);

    template <typename C>
    friend std::ostream &operator<<(std::ostream &out, const
BinarySearchTree<C> &bTree);

    //для задания:
    void simplifyTheFormula();
    BinarySearchTree(const std::string &str);
};

template <typename T>
void BinarySearchTree<T>::deleteSpaces(std::string &str)
{
    str.erase(std::remove_if(str.begin(), str.end(), []
(unsigned char sym) { return std::isspace(sym); }),
str.end());
}

template <typename T>
BinarySearchTree<T>::BinarySearchTree(const
BinarySearchTree<T> &tree) : head(this-
>recursiveCopy(tree.head, nullptr)) {}

template <typename T>
BinarySearchTree<T> &BinarySearchTree<T>::operator=(const
BinarySearchTree<T> &tree)
{
    if (this != &tree)
```

```
        this->head = this->recursiveCopy(tree.head, nullptr);
    return *this;
}

template <typename T>
std::shared_ptr<BinarySearchTreeNode<T>>
BinarySearchTree<T>::recursiveCopy(const
std::shared_ptr<BinarySearchTreeNode<T>> &ptrNodeToCopy,
const std::shared_ptr<BinarySearchTreeNode<T>> &ptrParent)
{
    if (ptrNodeToCopy != nullptr)
    {
        std::shared_ptr<BinarySearchTreeNode<T>> newObj(new
BinarySearchTreeNode<T>(ptrNodeToCopy->obj, ptrParent));
        newObj->left = recursiveCopy(ptrNodeToCopy->left,
newObj);
        newObj->right = recursiveCopy(ptrNodeToCopy->right,
newObj);
        return newObj;
    }
    else
        return nullptr;
}

template <typename T>
BinarySearchTree<T>::BinarySearchTree(BinarySearchTree<T>
&&tree) : head(std::move(tree.head)) {}

template <typename T>
BinarySearchTree<T>
&BinarySearchTree<T>::operator=(BinarySearchTree<T> &&tree)
{
    if (&tree != this)
        this->head = std::move(tree.head);
    return *this;
}

template <typename T>
void BinarySearchTree<T>::remove(const T &val)
{
    this->recursiveRemove(val, this->head);
}

template <typename T>
void BinarySearchTree<T>::recursiveRemove(const T &val,
std::shared_ptr<BinarySearchTreeNode<T>> &ptrNode, const
std::shared_ptr<BinarySearchTreeNode<T>> &ptrParent)
{
    if (ptrNode != nullptr)
    {
        if (ptrNode->obj < val)
            recursiveRemove(val, ptrNode->left, ptrNode);
        else if (ptrNode->obj > val)
            recursiveRemove(val, ptrNode->right, ptrNode);
        else
```

```
        {
            std::shared_ptr<BinarySearchTreeNode<T>> ptrMin =
nullptr;
            if (ptrNode->right != nullptr)
            {
                ptrMin = this->min(ptrNode->right);
                ptrMin->left = ptrNode->left;
                if (ptrMin->left != nullptr)
                    ptrMin->left->parent = ptrMin;
                if (ptrMin == ptrNode->right)
                    ptrMin->right = nullptr;
                else
                {
                    ptrMin->right = ptrNode->right;
                    ptrMin->right->parent = ptrMin;
                }
            }
            else if (ptrNode->left != nullptr)
            {
                ptrMin = ptrNode->left;
                ptrNode->left->parent = ptrMin;
            }
            if (ptrMin != nullptr)
            {
                if (ptrMin != ptrNode->right)
                    ptrMin->parent.lock()->left = nullptr;
                ptrMin->parent = ptrParent;
            }
            if (ptrParent != nullptr)
            {
                if (ptrParent->left == ptrNode)
                    ptrParent->left = ptrMin;
                else
                    ptrParent->right = ptrMin;
            }
            else
                this->head = ptrMin;
        }
    }
}

template <typename T>
std::shared_ptr<BinarySearchTreeNode<T>>
BinarySearchTree<T>::min(const
std::shared_ptr<BinarySearchTreeNode<T>> &ptrNode)
{
    std::shared_ptr<BinarySearchTreeNode<T>> ptr = ptrNode;
    if (ptrNode != nullptr)
    {
        while (ptr->left != nullptr)
            ptr = ptr->left;
    }
    return ptr;
}
```

```
template <typename T>
void BinarySearchTree<T>::insert(const T &val)
{
    this->recursiveInsert(val, this->head);
}

template <typename T>
void BinarySearchTree<T>::recursiveInsert(const T &val,
std::shared_ptr<BinarySearchTreeNode<T>> &ptrNode, const
std::shared_ptr<BinarySearchTreeNode<T>> &ptrParent)
{
    if (ptrNode == nullptr)
        ptrNode =
std::shared_ptr<BinarySearchTreeNode<T>>(new
BinarySearchTreeNode<T>(val, ptrParent));
    else if (val > ptrNode->obj)
        this->recursiveInsert(val, ptrNode->right, ptrNode);
    else if (val < ptrNode->obj)
        this->recursiveInsert(val, ptrNode->left, ptrNode);
}

template <typename T>
std::ostream &operator<<(std::ostream &out, const
BinarySearchTree<T> &bTree)
{
    std::vector<T> vectorOfAtoms = bTree.prefixTraverse();
    for (auto it = vectorOfAtoms.begin(); it !=
vectorOfAtoms.end(); ++it)
    {
        if (it == vectorOfAtoms.begin())
            out << (*it);
        else
            out << ' ' << (*it);
    }
    return out;
}

template <typename T>
BinarySearchTree<T>::BinarySearchTree()
{
    this->head =
std::unique_ptr<BinarySearchTreeNode<T>>(nullptr);
}

template class BinarySearchTree<char>;

template <>
void
BinarySearchTree<char>::recursiveSimplifyTheFormula(std::shar
ed_ptr<BinarySearchTreeNode<char>> &ptrNode)
{
    if (ptrNode != nullptr)
    {
        if (ptrNode->obj == '+')
        {
```

```
            ptrNode->obj = '*';
            ptrNode->left->obj = '2';
            ptrNode->left->left = nullptr;
            ptrNode->left->right = nullptr;
        }
        else
            recursiveSimplifyTheFormula(ptrNode->left);
        recursiveSimplifyTheFormula(ptrNode->right);
    }
}

template <>
void BinarySearchTree<char>::simplifyTheFormula()
{
    this->recursiveSimplifyTheFormula(this->head);
}

template <typename T>
void BinarySearchTree<T>::simplifyTheFormula()
{
    try
    {
        throw MyException(ExceptionsNames::ex_logic_error,
"Using method with unsupported type");
    }
    catch (const std::exception &e)
    {
        std::cerr << e.what() << '\n';
    }
}

template <>
bool BinarySearchTree<char>::checkTaskBin(const
std::shared_ptr<BinarySearchTreeNode<char>> &ptrNode)
{
    if (ptrNode != nullptr)
    {
        if (ptrNode->obj == '-' || ptrNode->obj == '+' ||
ptrNode->obj == '*' || ptrNode->obj == '/')
            return (ptrNode->left != nullptr && ptrNode-
>right != nullptr) ? checkTaskBin(ptrNode->left) &&
checkTaskBin(ptrNode->right) : 0;
        else if ((ptrNode->obj >= '0' && ptrNode->obj <= '9')
|| (ptrNode->obj >= 'a' && ptrNode->obj <= 'z'))
            return (ptrNode->left == nullptr && ptrNode-
>right == nullptr) ? 1 : 0;
        else
            return 0;
    }
    else
        return 1;
}

template <>
```

```
std::shared_ptr<BinarySearchTreeNode<char>>
BinarySearchTree<char>::recursiveConstructorFromStr(const
std::string &str, std::shared_ptr<BinarySearchTreeNode<char>>
&ptrNode, const std::shared_ptr<BinarySearchTreeNode<char>>
&ptrParent)
{
    char curObj = str[curStrIndex];
    ptrNode = std::shared_ptr<BinarySearchTreeNode<char>>(new
BinarySearchTreeNode<char>(curObj, ptrParent));
    if (curStrIndex >= str.size())
        return nullptr;
    ++curStrIndex;
    if (curObj == '-' || curObj == '+' || curObj == '*' ||
curObj == '/')
    {
        ptrNode->left = recursiveConstructorFromStr(str,
ptrNode->left, ptrNode);
        ptrNode->right = recursiveConstructorFromStr(str,
ptrNode->right, ptrNode);
    }
    return ptrNode;
}

template <>
BinarySearchTree<char>::BinarySearchTree(const std::string
&str)
{
    std::string newStr = str;
    if (!newStr.empty())
        this->deleteSpaces(newStr);
    this->head = this->recursiveConstructorFromStr(newStr,
this->head);
    try
    {
        if (!this->checkTaskBin(this->head) || curStrIndex <
newStr.size())
            throw
MyException(ExceptionsNames::ex_syntax_error, "Wrong
string");
    }
    catch (const std::exception &e)
    {
        this->head = nullptr;
        std::cerr << e.what() << '\n';
    }
}

template <typename T>
BinarySearchTree<T>::BinarySearchTree(const std::string &str)
{
    try
    {
        throw MyException(ExceptionsNames::ex_logic_error,
"Using method with unsupported type");
    }
```

```
    catch (const std::exception &e)
    {
        std::cerr << e.what() << '\n';
    }
}

template <typename T>
std::vector<T> BinarySearchTree<T>::prefixTraverse() const
{
    std::vector<T> vectorOfNodes;
    recursivePrefixTraverse(vectorOfNodes, this->head);
    return vectorOfNodes;
}

template <typename T>
std::vector<T> BinarySearchTree<T>::postfixTraverse() const
{
    std::vector<T> vectorOfNodes;
    recursivePostfixTraverse(vectorOfNodes, this->head);
    return vectorOfNodes;
}

template <typename T>
std::vector<T> BinarySearchTree<T>::infixTraverse() const
{
    std::vector<T> vectorOfNodes;
    recursiveInfixTraverse(vectorOfNodes, this->head);
    return vectorOfNodes;
}

template <typename T>
void
BinarySearchTree<T>::recursivePrefixTraverse(std::vector<T>
&vectorOfNodes, const
std::shared_ptr<BinarySearchTreeNode<T>> &nodePtr) const
{
    if (nodePtr != nullptr)
    {
        vectorOfNodes.push_back(nodePtr->obj);
        this->recursivePrefixTraverse(vectorOfNodes, nodePtr-
>left);
        this->recursivePrefixTraverse(vectorOfNodes, nodePtr-
>right);
    }
}

template <typename T>
void
BinarySearchTree<T>::recursivePostfixTraverse(std::vector<T>
&vectorOfNodes, const
std::shared_ptr<BinarySearchTreeNode<T>> &nodePtr) const
{
    if (nodePtr != nullptr)
    {
```

```
        this->recursivePostfixTraverse(vectorOfNodes,
nodePtr->left);
        this->recursivePostfixTraverse(vectorOfNodes,
nodePtr->right);
        vectorOfNodes.push_back(nodePtr->obj);
    }
}

template <typename T>
void
BinarySearchTree<T>::recursiveInfixTraverse(std::vector<T>
&vectorOfNodes, const
std::shared_ptr<BinarySearchTreeNode<T>> &nodePtr) const
{
    if (nodePtr != nullptr)
    {
        this->recursiveInfixTraverse(vectorOfNodes, nodePtr-
>left);
        vectorOfNodes.push_back(nodePtr->obj);
        this->recursiveInfixTraverse(vectorOfNodes, nodePtr-
>right);
    }
}

template <typename T>
bool BinarySearchTree<T>::empty() const
{
    return (this->head == nullptr) ? 0 : 1;
}

#endif //!__BINARYSEARCHTREE__H__
```

MyException.h

```
#ifndef MYEXS_H
#define MYEXS_H

#include <ostream>
#include <string>
#include <stdexcept>

enum class ExceptionsNames
{
    ex_logic_error,
    ex_syntax_error,
};

class MyException : public std::runtime_error
{
    std::string ex_message;

public:
    MyException(const ExceptionsNames &inputed_ex_name, const
std::string &str);
```

```
    friend std::ostream &operator<<(std::ostream &out, const
MyException &obj);
};
#endif
```

MyException.cpp

```
#include "../libs/MyException.h"

MyException::MyException(const ExceptionsNames
&inputed_ex_name, const std::string &str) :
std::runtime_error(str)
{
    switch (inputed_ex_name)
    {
        case ExceptionsNames::ex_logic_error:
            this->ex_message = "Logic Error: ";
        break;
        case ExceptionsNames::ex_syntax_error:
            this->ex_message = "Syntax Error: ";
        break;
    }
    this->ex_message += str;
}

std::ostream &operator<<(std::ostream &out, const MyException
&obj)
{
    out << obj.ex_message;
    return out;
}
```

Makefile

```
compiler = g++
flags = -c -g -std=c++17 -Wall
appname = lab3
lib_dir = Sources/libs/
src_dir = Sources/srcs/

src_files := $(wildcard $(src_dir)*)
obj_files := $(addsuffix .o, $(basename $(notdir $
(src_files))))

define compile
    $(compiler) $(flags) $<
endef

programbuild: $(obj_files)
    $(compiler) $^ -o $(appname)

%.o: $(src_dir)/%.cpp $(lib_dir)/*.h
```

```
    $(call compile)

clean:
    rm -f *.o $(appname)
```