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

ОТЧЕТ

по лабораторной работе №2

по дисциплине «Алгоритмы и структуры данных»

Тема: Иерархические списки

Студент гр. 9304 _____ Афанасьев А.

Преподаватель _____ Филатов А. Ю.

Санкт-Петербург

2020

Цель работы.

Ознакомиться с понятием иерархического списка, реализовать его и решить поставленную задачу с его помощью.

Постановка задачи.

Вариант 18.

Пусть логическое выражение представлено иерархическим списком. В выражение входят операции, константы и переменные, которые являются атомами списка.

Операции представляются в префиксной форме: (операция аргументы). Аргументов может быть 2 или 1. Требуется добавить четвертую операцию с 2 аргументами. По умолчанию есть NOT, OR и AND.

В задании требуется вычислить значение поданного в скобочной записи логического выражения.

Дополнительно задаётся список значений переменных вида:

((x1 c1) (x2 c2) ... (xk ck)),

где xi – переменная, а ci – её значение.

Выполнение работы:

Программа на вход принимает две строки — два иерархических списка, записанных в скобочном виде. Первой строкой принимается логическое выражение, второй — значения переменных. Пример изображен на рисунке 1, визуализация первой строки на рисунке 2, а второй строки на рисунке 3.

```
./lab2 "(XOR A (OR  2 (AND D (NOT G))))" "((A TRUE)(D 1)(G 0))"
```

Рисунок 1 - Пример самостоятельного запуска программы

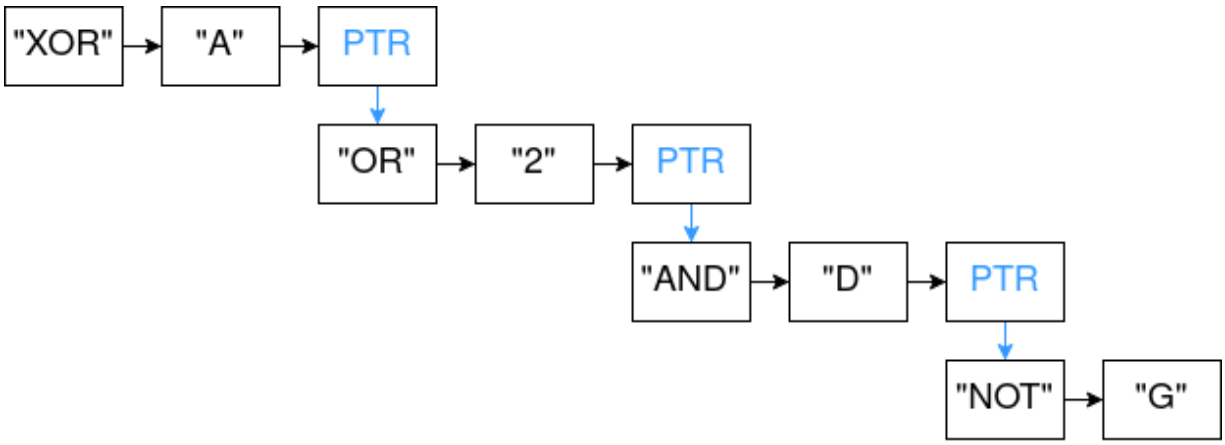

Рисунок 2 — Визуализация первой строки из примера

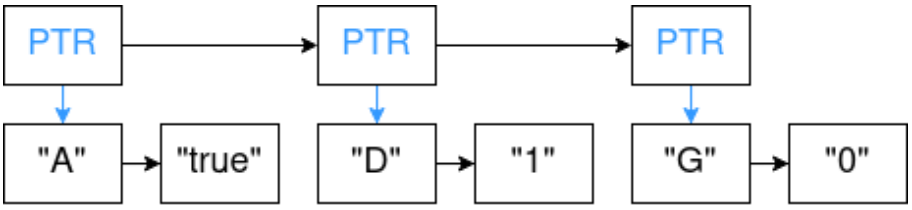

Рисунок 3 - Визуализация второй строки из примера

Структура Node имеет всего два поля: std::variant item, в котором может находиться std::string или умный указатель на MyList, и умный указатель next на следующий элемент иерархического списка. Также его другом является класс MyList.

Класс MyList хранит в своих полях только лишь умный указатель на голову иерархического списка. Метод calculateLogicExpr() выполняет вычисление логического выражения, хранящегося в this. Для этого метода реализовано несколько вспомогательных приватных методов: isNum() - проверяет, является ли строка записью числа, calculate() - метод, рекурсивно вызывающий самого себя для вычисления логического выражения, первый раз он запускается в методе calculateLogicExpr(), findValue() - метод, возвращающий значение переданной в него переменной. Для конструирования объекта класса был перегружен конструктор, который, используя публичный метод append(), рекурсивно, распознавая скобочную запись списка, конструирует объект.

Класс MyException был создан для обработки синтаксических ошибок ввода, он в зависимости от типа ошибки выдает пользователю соответствующее сообщение о ней. Сами исключения бросаются в процессе вычисления логического выражение и конструирования объекта.

Метод calculateLogicExpr(), а точнее метод calculate(), так как первый метод всего лишь ловит исключения последнего, принимает на вход аргумент типа умный указатель на Node, иерархический список значений переменных и особый флаг CheckArgumentFlag, о котором позже. Алгоритм проверяет, что находится в объекте Node, указатель на который ему передали. Если это AND, то calculate() возвращает результат логического И от себя, вызванного от следующего атома, и от себя, вызванного от второго следующего атома. Аналогично для OR, NOT и XOR (та операция, которую добавили по условию). Если в указателе оказалась, не операция, а число, то алгоритм просто вернет это число. Если это переменная, то алгоритм вернет ее значение из списка значений переменных с помощью метода findValue(). А если это указатель на список, то для алгоритм вернет значение, которое вернется от вызова calculate() от головы этого списка.

Теперь про флаг CheckArgumentFlag. Он нужен, чтобы проверять отсутствие или излишек аргументов. Например, для первого аргумента OR/XOR/AND нужно проверить, что после него есть еще второй аргумент, в противном случае вызывается исключение. Для последнего аргумента наоборот — нужно проверить, что после него нет еще одного аргумента.

Метод findValue() просто проходится от первого атома списка значений переменных, пока тот не закончится (в таком случае вызывается исключение) или пока он не найдет значение переменной-аргумента. Синтаксические ошибки списка значений переменных также обрабатываются исключениями.

Тестирование.

Программа собирается через Makefile командой make, после чего создатся файл lab2. Программе при запуске можно передавать логическое

выражение и список значений переменных. Обе строки должны быть записаны в скобочном виде. Пример запуска можно посмотреть на рисунке 1.

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 4. А в таблице 1 можно посмотреть примеры строк-тестов.

```
[redacted]$ python testScript.py
Make sure that this script is in the same directory as the program execute file.

test0:
Input: "(AND A(OR f (NOT false)))" "((A 0)(f FALSE))"
CorrectAnswer: 0
Answer: 0
Result: success

Total: Successes: 1. Fails: 0
```

Рисунок 4 - Пример работы тестирующего скрипта

Таблица 1 – Тесты и их результаты

№	Входные данные	Выход
0	"(AND A(OR f (NOT false)))" "((A 0)(f FALSE))"	0
1	"(XOR A (OR 2 (AND D (NOT G))))" "((A TRUE)(D 1)(G 0))"	0
2	"(XOR 1 0)" "()"	1
3	"(AND 1 0)" "()"	0
4	"(OR 1 A)" "((A FALSE))"	1
5	"(TRUE)" "()"	1
6	"(NOT K)" "((K FALSE))"	1
7	"(NOT (AND K (XOR FALSE (OR L Y))))" "((K 1) (L FALSE) (Y 1))"	0
8	"(XOR TruE (NOT (AND g w)))" "((TruE 1)(g 0)(w 1))"	0
9	"(AND (OR (NOT A) B) (XOR (AND C false) true))" "((A 1) (B 1) (C 1))"	1

Комментарий к 8 тесту:

Запись TruE не является валидной константной и распознается как переменная. Валидными константами являются, например, 12321, TRUE и false.

Вывод.

В ходе выполнения работы ознакомились со структурой иерархического списка и его особенностями. Также познакомились с умными указателями и std::variant.

Была разработана программа, создающая иерархический список и вычисляющая логические выражение с его помощью. Выяснили, что с точки зрения экономии памяти, использование иерархического списка для выполнения поставленной задачи не сильно оправдано, так как при

конструировании иерархического списка и вычислении логического выражения с его помощью используется рекурсивная обработка самого иерархического списка. Весь исходный код можно посмотреть в приложении А.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД

main.h.

```
#ifndef MAIN_H
#define MAIN_H

#include "MyList.h"
#include <iostream>

#endif
```

main.cpp.

```
#include "../libs/main.h"
int main(int n, char **args)
{
    try
    {
        MyList listValues(args[2]);
        MyList listName(args[1]);
        std::cout <<
listName.calculateLogicExpr(listValues);
    }
    catch (const MyException &e)
    {
        std::cout << e << '\n';
    }
    return 0;
}
```

MyList.h.

```
#ifndef MYLIST_H
#define MYLIST_H

#include <variant>
#include <string>
#include "MyException.h"
#include <memory>
#include <iostream>

enum class CheckArgumentFlag
{
    Start,
    NextIsArg,
    NextIsNothing
};

class MyList;

struct Node
```

```
{
private:
    friend class MyList;
    std::variant<std::string, std::shared_ptr<MyList>>
item;
    std::shared_ptr<Node> next = nullptr;
};

class MyList
{
    std::shared_ptr<Node> head = nullptr;
    static const bool isNum(const std::string_view
&str);
    static const bool calculate(std::shared_ptr<Node>
ptr, const MyList &listOfValues, const
CheckArgumentFlag check);
    static const long long findValue(const
std::string_view &str, const MyList &list);

public:
    MyList(const std::string_view &&str);
    const bool calculateLogicExpr(const MyList
&listOfValues);
    void append(const Node &item);
};

#endif
```

MyList.cpp

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

MyList::MyList(const std::string_view &&str)
{
    bool isFirstBar = 1;
    size_t index = 0;
    while (index < str.length())
    {
        if (str[index] == '(')
        {
            size_t leftIndex = index;
            long long bar = 0;
            do
            {
                if (str[index] == '(')
                    --bar;
                else if (str[index] == ')')
                    ++bar;
                ++index;
            } while (bar != 0 && index < str.length());
            if (bar != 0)
                throw
MyException(ExceptionsNames::ex_list_is_wrong);
            if (isFirstBar == 1)
            {
```

```
                isFirstBar = 0;
                index = leftIndex + 1;
            }
            else
            {
                std::shared_ptr<MyList> newItem(new
MyList(str.substr(leftIndex, index - leftIndex)));
                Node newNode;
                newNode.item = newItem;
                this->append(newNode);
            }
        }
        else if (str[index] >= 'A' && str[index] <= 'Z'
|| str[index] >= 'a' && str[index] <= 'z' || str[index]
>= '0' && str[index] <= '9' || str[index] == '-')
        {
            size_t leftIndex = index;
            index =
str.find_first_not_of("0123456789QWERTYUIOPASDFGHJKLZXC
VBNMqwertyuiopasdfghjklzxcvbnm-", index + 1);
            const std::string_view newItem =
str.substr(leftIndex, index - leftIndex);
            Node newNode;
            newNode.item = (std::string)newItem;
            this->append(newNode);
        }
        else if (str[index] == ' ' || str[index] == '\
t' || str[index] == ')')
            ++index;
        else
            throw
MyException(ExceptionsNames::ex_list_is_wrong);
    }
}

void MyList::append(const Node &newNode)
{
    std::shared_ptr<Node> node =
std::make_shared<Node>(newNode);
    std::shared_ptr<Node> ptr = this->head;
    if (this->head != nullptr)
    {
        while (ptr->next != nullptr)
            ptr = ptr->next;
        ptr->next = node;
    }
    else
        this->head = node;
}

const bool MyList::isNum(const std::string_view &str)
{
    size_t i = 0;
    if (str[0] == '-')
        i = 1;
    for (; i < str.length(); ++i)
```

```
        if (!std::isdigit(str[i]))
            return 0;
    return 1;
}

const bool MyList::calculateLogicExpr(const MyList
&listOfValues)
{
    try
    {
        return MyList::calculate(this->head,
listOfValues, CheckArgumentFlag::Start);
    }
    catch (const MyException &ex)
    {
        std::cout << ex << '\n';
        return 0;
    }
}

const bool MyList::calculate(std::shared_ptr<Node>
ptrNode, const MyList &listOfValues, const
CheckArgumentFlag check)
{
    if (ptrNode == nullptr)
        throw
MyException(ExceptionsNames::ex_expr_is_wrong);
    if (check == CheckArgumentFlag::NextIsArg &&
ptrNode->next == nullptr)
        throw
MyException(ExceptionsNames::ex_argument_doesnt_exists)
;
    if (check == CheckArgumentFlag::NextIsNothing &&
ptrNode->next != nullptr)
        throw
MyException(ExceptionsNames::ex_unnecessary_argument);

    else if
(std::holds_alternative<std::string>(ptrNode->item))
    {
        std::string str =
std::get<std::string>(ptrNode->item);
        if (str == "NOT")
            return !MyList::calculate(ptrNode->next,
listOfValues, CheckArgumentFlag::NextIsNothing);
        else if (str == "OR")
        {
            const bool left =
MyList::calculate(ptrNode->next, listOfValues,
CheckArgumentFlag::NextIsArg);
            const bool right =
MyList::calculate(ptrNode->next->next, listOfValues,
CheckArgumentFlag::NextIsNothing);
            return left || right;
        }
        else if (str == "XOR")
```

```
        {
            const bool left =
MyList::calculate(ptrNode->next, listOfValues,
CheckArgumentFlag::NextIsArg);
            const bool right =
MyList::calculate(ptrNode->next->next, listOfValues,
CheckArgumentFlag::NextIsNothing);
            return !left && right || left && !right;
        }
        else if (str == "AND")
        {
            const bool left =
MyList::calculate(ptrNode->next, listOfValues,
CheckArgumentFlag::NextIsArg);
            const bool right =
MyList::calculate(ptrNode->next->next, listOfValues,
CheckArgumentFlag::NextIsNothing);
            return left && right;
        }
        else
        {
            if (check == CheckArgumentFlag::Start &&
ptrNode->next != nullptr)
                throw
MyException(ExceptionsNames::ex_unnecessary_argument);
            if (isNum(str))
                return std::stoll(str);
            else if (str == "true" || str == "TRUE")
                return 1;
            else if (str == "false" || str == "FALSE")
                return 0;
            else
                return findValue(str, listOfValues);
        }
    }
    else
        return
MyList::calculate(std::get<std::shared_ptr<MyList>>(ptr
Node->item)->head, listOfValues,
CheckArgumentFlag::Start);
}

const long long MyList::findValue(const
std::string_view &aim, const MyList &list)
{
    if (list.head == nullptr)
        throw
MyException(ExceptionsNames::ex_empty_values_list);
    std::shared_ptr<Node> ptr = list.head;
    while (ptr != nullptr)
    {
        if (!
std::holds_alternative<std::shared_ptr<MyList>>(ptr-
>item))
            throw
MyException(ExceptionsNames::ex_incorrect_values_list);
```

```
        std::shared_ptr<MyList> ptrList =
std::get<std::shared_ptr<MyList>>(ptr->item);
        if (ptrList->head == nullptr || ptrList->head-
>next == nullptr || !
std::holds_alternative<std::string>(ptrList->head-
>item) || !std::holds_alternative<std::string>(ptrList-
>head->next->item) || ptrList->head->next->next !=
nullptr)
            throw
MyException(ExceptionsNames::ex_incorrect_values_list);

        if (aim == std::get<std::string>(ptrList->head-
>item))
        {
            std::string rightStr =
std::get<std::string>(ptrList->head->next->item);
            if (rightStr == "TRUE" || rightStr ==
"true")
                return 1;
            else if (rightStr == "FALSE" || rightStr ==
"false")
                return 0;
            else if (isNum(rightStr))
                return std::stoll(rightStr);
            else
                throw
MyException(ExceptionsNames::ex_incorrect_values_list);
        }
        ptr = ptr->next;
    }
    throw
MyException(ExceptionsNames::ex_the_value_of_the_variab
le_is_missing);
}
```

MyException.h.

```
#ifndef MYEXS_H
#define MYEXS_H

#include <ostream>
#include <string>

enum class ExceptionsNames
{
    ex_expr_is_wrong,
    ex_list_is_wrong,
    ex_empty_values_list,
    ex_incorrect_values_list,
    ex_argument_doesnt_exists,
    ex_unnecessary_argument,
    ex_the_value_of_the_variable_is_missing
};

class MyException
```

```
{
    std::string ex_message;

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

MyException.cpp.

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

MyException::MyException(const ExceptionsNames
&inputed_ex_name)
{
    this->ex_message = "Syntax error: ";
    switch (inputed_ex_name)
    {
    case ExceptionsNames::ex_expr_is_wrong:
        this->ex_message += "The inputted expression
is wrong";
        break;
    case ExceptionsNames::ex_argument_doesnt_exists:
        this->ex_message += "One of the arguments
doesn't exists";
        break;
    case ExceptionsNames::ex_empty_values_list:
        this->ex_message += "The inputted list of
variable values is empty";
        break;
    case ExceptionsNames::ex_incorrect_values_list:
        this->ex_message += "The inputted list of
variable values is wrong";
        break;
    case ExceptionsNames::ex_unnecessary_argument:
        this->ex_message += "One of the arguments is
unnecessary";
        break;
    case
ExceptionsNames::ex_the_value_of_the_variable_is_missi
ng:
        this->ex_message += "The value of one of the
variables is missing";
        break;
    case ExceptionsNames::ex_list_is_wrong:
        this->ex_message += "The list is wrong";
        break;
    }
}

std::ostream &operator<<(std::ostream &out, const
MyException &obj)
{
```

```
    out << obj.ex_message;
    return out;
}
```

Makefile.

```
compiler = g++
flags = -c -g -std=c++17
appname = lab2
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)
```