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

ОТЧЕТ

по лабораторной работе №1

по дисциплине «Алгоритмы и структуры данных»

Тема: Рекурсия

Вариант 9

Студент гр. 8304 Сани З. Б.

Преподаватель Фирсов М.А.

Санкт-Петербург
2019

Цель работы.

Ознакомиться с основными понятиями и приёмами рекурсивного программирования, получить навыки программирования рекурсивных процедур и функций на языке программирования C++.

Постановка задачи.

1) проанализировать полученное задание, выделив рекурсивно определяемые информационные объекты и (или) действия;
2) разработать программу, использующую рекурсию;
3) сопоставить рекурсивное решение с итеративным решением задачи;
4) сделать вывод о целесообразности и эффективности рекурсивного решения данной задачи.

9 Разработать программу, которая по заданному простому_логическому выражению (определение понятия см. в предыдущей задаче), не содержащему вхождений простых идентификаторов, вычисляет значение этого выражения.

Описание алгоритма.

Чтобы вычислить значение выражения с помощью рекурсии, вы должны разделить выражение на три категории (первое выражение + знак + второе выражение) в случае ,когда перед логикой есть скобка ,затем проверить каждое выражение и затем отрицать значение выражения, где есть оператор NOT, иначе рекурсивно вычислить значение этого выражения. Оператор NOT также рекурсивно определяет, какую логику следует отрицать. Затем после рекурсивного разбиения двух выражений вы вычисляете оба и возвращаете значение.

ALGORITHM:

```
calculate_expression(input, position)
        if input = “TRUE”
            return true
        if input = “FALSE”
            return false
        if input = “NOT(TRUE)”
            return false
        if input = “NOT(FALSE)”
            return true

    if input[position] = “(“
            expr1 = is a substring of input
            expr2 =  is a substring of input
            sign  = is a substring of input
            if expr1 contains Not statement
                    expr1= calculate_expression(!expr1,0)
            if expr2 contains Not statement
                    expr2= calculate_expression(!expr2,0)
            a = calculate_expressionexpr1, 0)
             b = calculate_expression(expr2, 0)
            return a sign b;
```

Спецификация программы.

Программа предназначена для синтаксического анализа выражения методом рекурсии.

Программа написана на языке C++. Входными данными является строка из консоли или строка из файла, как указано, но пользователь. Выходными данными являются промежуточные значения проверки выражения который является либо TRUE / FALSE или сообщение об ошибке, когда выражение не является допустимым

ПРИЛОЖЕНИЕ.

1) ТЕСТИРОВАНИЕ:

Работа программы для строки(TRUE AND NOT(FALSE OR NOT(TRUE AND FALSE)))

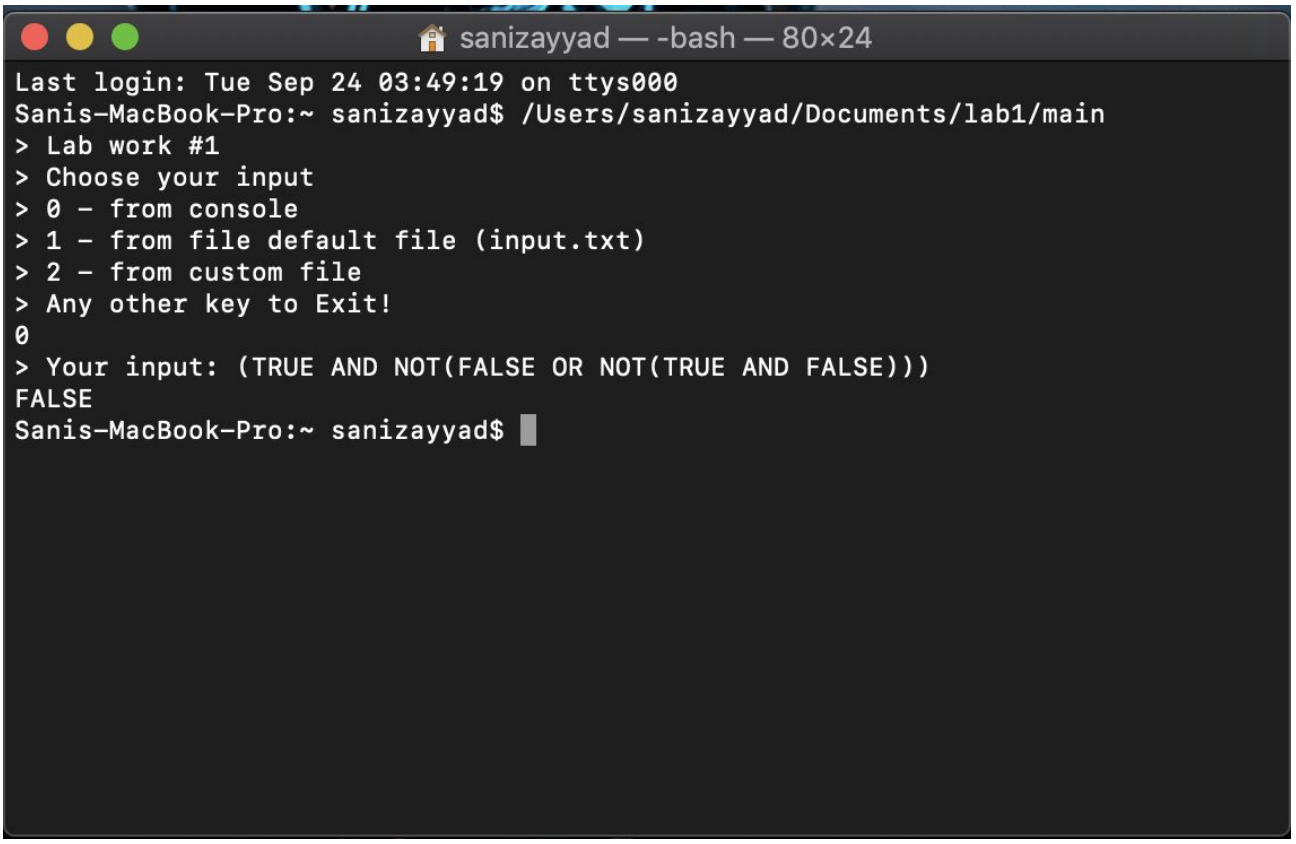
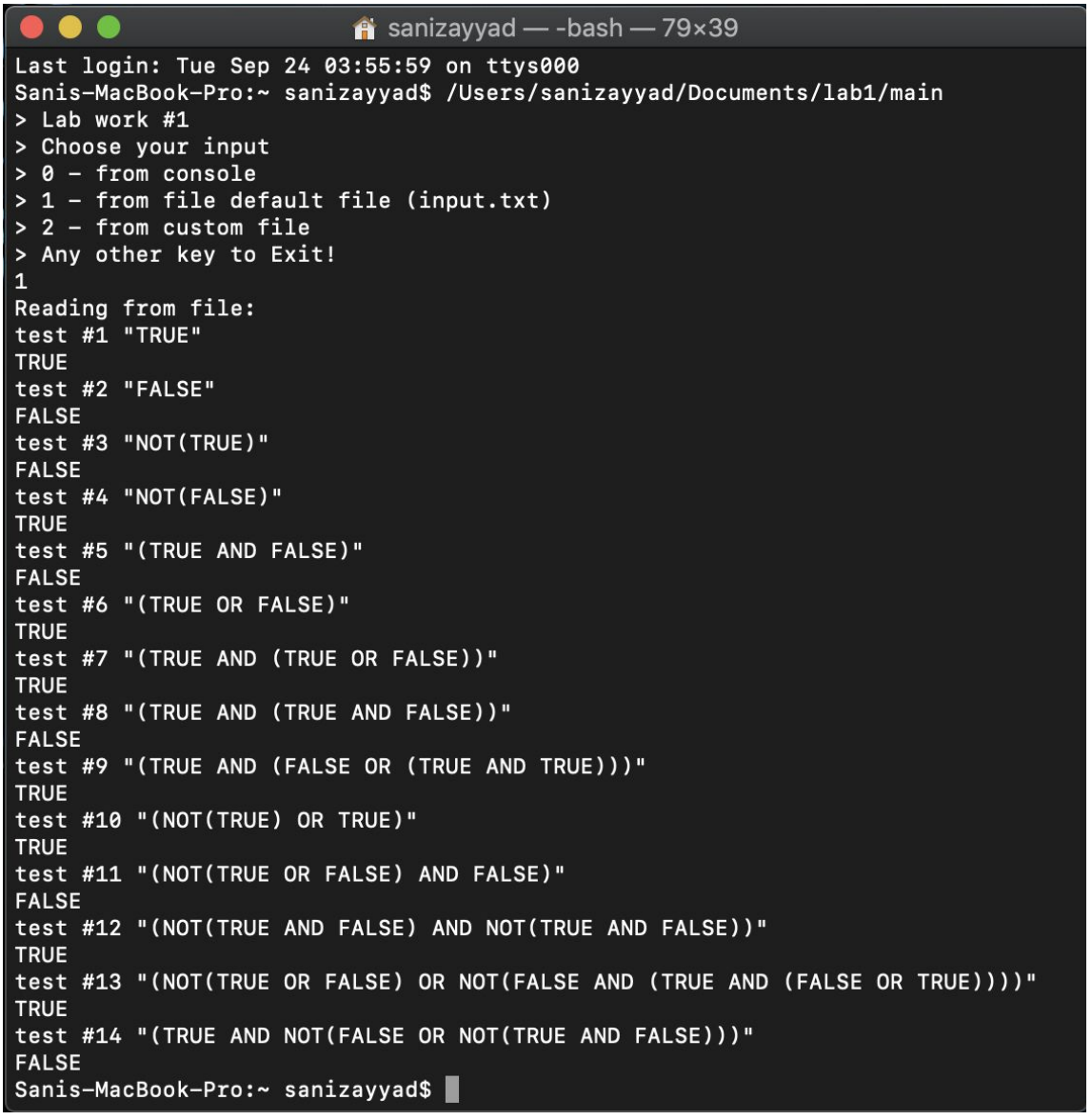


Таблица результатов ввода/вывода тестирования программы

Входная строка	Вывод программы
TRUE	TRUE
FALSE	FALSE
NOT(TRUE)	FALSE
NOT(FALSE)	TRUE
(TRUE AND FALSE)	FALSE
(TRUE OR FALSE)	TRUE
(TRUE AND (TRUE OR FALSE))	TRUE

(TRUE AND (TRUE AND FALSE))	FALSE
(TRUE AND (FALSE OR (TRUE AND TRUE)))	TRUE
(NOT(TRUE) OR TRUE)	TRUE
(NOT(TRUE OR FALSE) AND FALSE)	FALSE
(NOT(TRUE AND FALSE) AND NOT(TRUE AND FALSE))	TRUE
(NOT(TRUE OR FALSE) OR NOT(FALSE AND (TRUE AND (FALSE OR TRUE))))	TRUE
(TRUE AND NOT(FALSE OR NOT(TRUE AND FALSE)))	FALSE



Выводы.

За время работы я познакомился с рекурсией, научился работать с этим методом. Я считаю, что метод рекурсии эффективен для этой задачи вычисления значения выражения .

2) ИСХОДНЫЙ КОД:

```
#include <iostream>
#include <string>
#include <fstream>
using namespace std;

//declaration of Functions
void ProceedInput();
string ReadFromConsole();
void ReadFromFile(string filename);
bool checkValidity(string str);
string notFunction(string str);
bool expression(string str, int pointer);
void calculate(string input);

//this is the starting point of the progran
// it asks user to select input option and uses the input to calculate the logic
void ProceedInput()
{
   //dialog
   cout << "> Lab work #1" << endl;
   cout << "> Choose your input" << endl;
   cout << "> 0 - from console" << endl;
   cout << "> 1 - from file default file (input.txt)" << endl;
   cout << "> 2 - from custom file" << endl;
   cout << "> Any other key to Exit!" << endl;

   int command = 0;
   //command input
   cin >> command;
   cin.ignore();

   //default test file path
   string FilePath = "/Users/sanizayyad/Documents/input.txt";
   string input;

   // I used switch statement to trigger the action base on the dialog
   switch (command)
   {
   case 0:
```

```
        cout << "> Your input: ";
        input = ReadFromConsole();
        calculate(input);
        break;
    case 1:
        ReadFromFile(FilePath);
        break;
    case 2:
        cout << "> FilePath: ";
        cin >> FilePath;
        ReadFromFile(FilePath);
        break;
    default:
        cout << "GOODBYE!";
    }
}
// this function basically just get input from the console
string ReadFromConsole()
{
    string input;
    getline(cin, input);
    return input;
}
//this function iterates the lines of the file and trigger calculate on every line as input
void ReadFromFile(string filename)
{
    ifstream file;
    file.open(filename);

    if (file.is_open())
    {
        cout << "Reading from file:"
             << "\n";

        int super_count = 0;

        while (!file.eof())
        {
            super_count++;
            string str;
            getline(file, str);
            cout << "test #" << super_count << " \"" + str + "\""
                 << "\n";
            calculate(str);
        }
    }
    else
    {
        cout << "File not opened";
    }
```

```
        file.close();
    }
    // checking if the brackets are balanced
    bool checkValidity(string str)
    {
        int balanceBracket = 0;
        int i = 0;
        for (i; i < str.size(); i++)
        {
            if (str[i] == ')')
                balanceBracket--;
            else if (str[i] == '(')
                balanceBracket++;
        }
        if (balanceBracket == 0)
            return true;
        return false;
    }
    //the main recursive function to check expression
    bool expression(string str, int pos)
    {
        if (str == "TRUE")
            return true;
        if (str == "FALSE")
            return false;
        if (str == "NOT(TRUE)")
            return false;
        if (str == "NOT(FALSE)")
            return true;

        if (str[pos] == '(')
        {
            pos++;
            bool a, b;
            string exp_1, exp_2, sign;

            //I splitted the expression three which means for every expression there's
            //expression1 Sign expression2//

            //checking the main demarcation between two expression
              //for example (TRUE AND FALSE) , here the first space after TRUE is our
demarcation
            int breakspace = 0;
            int bracket = 0;
            for (breakspace = pos; breakspace < str.size(); breakspace++)
            {
                if (str[breakspace] == ' ')
                {
                    if (bracket == 0)
                        break;
```

```
            }
            if (str[breakspace] == ')')
                bracket--;
            else if (str[breakspace] == '(')
                bracket++;
        }
        int tmp = breakspace - 1;

        //////expression 1
        exp_1 = str.substr(pos, tmp);
        //if there's a Not statement in expression1
        //we call notFunction to calculate the NOT(logic)
        if (exp_1.find("NOT") != string::npos)
        {
            exp_1 = notFunction(exp_1);
        }
        //////SIGN
        string tmpStr = str.substr(tmp + 2);
        int tmp_2 = tmpStr.find(" ");
        sign = tmpStr.substr(0, tmp_2);
        ////expression 2
        exp_2 = tmpStr.substr(tmp_2 + 1, tmpStr.size() - tmp_2 - 2);
        if (exp_2.find("NOT") != string::npos)
        {
            exp_2 = notFunction(exp_2);
        }

        //recursive on expression 1
        a = expression(exp_1, 0);
        // recursive on expression 2
        b = expression(exp_2, 0);

        // the  we return the outcome
        if (sign == "AND")
            return (a && b);
        else
            return (a || b);
    }
}
//this returns the negation of a logic.
string notFunction(string str)
{
    int tmp = str.find("NOT");
    string tmp_expression = str.substr(tmp + 3, str.size());
    int tmp_closing = tmp_expression.find(")");
    tmp_expression = tmp_expression.substr(0, tmp_closing);
    bool before;

    if (tmp_expression.find(" ") != string::npos)
    {
```

```
        before = expression(tmp_expression, 0);
    }
    else
    {
        tmp_expression = tmp_expression.substr(1, tmp_closing - 1);
        before = expression(tmp_expression, 0);
    }

    string notExp = before == true ? "NOT(TRUE)" : "NOT(FALSE)";
    return notExp;
}
void calculate(string input)
{
    if (checkValidity(input) && !input.empty())
    {
        if (expression(input, 0) == true)
            cout << "TRUE" << endl;
        else
            cout << "FALSE" << endl;
    }
    else
        cout << "\t ERROR!!!" << endl;
}

int main()
{
    //start
    ProceedInput();

    return 0;
}
```

```cpp
#include <iostream>
#include <string>
#include <fstream>
using namespace std;

//declaration of Functions
void ProceedInput();
string ReadFromConsole();
void ReadFromFile(string filename);
bool checkValidity(string str);
string notFunction(string str);
bool expression(string str, int pointer);
void calculate(string input);

//this is the starting point of the progran
// it asks user to select input option and uses the input to calculate the logic
void ProceedInput()
{
    //dialog
    cout << "> Lab work #1" << endl;
    cout << "> Choose your input" << endl;
    cout << "> 0 - from console" << endl;
    cout << "> 1 - from file default file (input.txt)" << endl;
    cout << "> 2 - from custom file" << endl;
    cout << "> Any other key to Exit!" << endl;

    int command = 0;
    //command input
    cin >> command;
    cin.ignore();

    //default test file path
    string FilePath = "/Users/sanizayyad/Documents/input.txt";
    string input;

    // I used switch statement to trigger the action base on the dialog
    switch (command)
    {
    case 0:
        cout << "> Your input: ";
        input = ReadFromConsole();
        calculate(input);
```

```
            break;
        case 1:
            ReadFromFile(FilePath);
            break;
        case 2:
            cout << "> FilePath: ";
            cin >> FilePath;
            ReadFromFile(FilePath);
            break;
        default:
            cout << "GOODBYE!";
        }
    }
    // this function basically just get input from the console
    string ReadFromConsole()
    {
        string input;
        getline(cin, input);
        return input;
    }
    //this function iterates the lines of the file and trigger calculate on every line
as input
    void ReadFromFile(string filename)
    {
        ifstream file;
        file.open(filename);

        if (file.is_open())
        {
            cout << "Reading from file:"
                 << "\n";

            int super_count = 0;

            while (!file.eof())
            {
                super_count++;
                string str;
                getline(file, str);
                cout << "test #" << super_count << " \"" + str + "\""
                     << "\n";
                calculate(str);
            }
```

```
        }
        else
        {
            cout << "File not opened";
        }
        file.close();
    }
    // checking if the brackets are balanced
    bool checkValidity(string str)
    {
        int balanceBracket = 0;
        int i = 0;
        for (i; i < str.size(); i++)
        {
            if (str[i] == ')')
                balanceBracket--;
            else if (str[i] == '(')
                balanceBracket++;
        }
        if (balanceBracket == 0)
            return true;
        return false;
    }
    //the main recursive function to check expression
    bool expression(string str, int pos)
    {
        if (str == "TRUE")
            return true;
        if (str == "FALSE")
            return false;
        if (str == "NOT(TRUE)")
            return false;
        if (str == "NOT(FALSE)")
            return true;

        if (str[pos] == '(')
        {
            pos++;
            bool a, b;
            string exp_1, exp_2, sign;

            //I splitted the expression three which means for every expression there's
            //expression1 Sign expression2//
```

```
            //checking the main demarcation between two expression
              //for example (TRUE AND FALSE) , here the first space after TRUE is our
demarcation
            int breakspace = 0;
            int bracket = 0;
            for (breakspace = pos; breakspace < str.size(); breakspace++)
            {
                if (str[breakspace] == ' ')
                {
                    if (bracket == 0)
                        break;
                }
                if (str[breakspace] == ')')
                    bracket--;
                else if (str[breakspace] == '(')
                    bracket++;
            }
            int tmp = breakspace - 1;

            //////expression 1
            exp_1 = str.substr(pos, tmp);
            //if there's a Not statement in expression1
            //we call notFunction to calculate the NOT(logic)
            if (exp_1.find("NOT") != string::npos)
            {
                exp_1 = notFunction(exp_1);
            }
            //////SIGN
            string tmpStr = str.substr(tmp + 2);
            int tmp_2 = tmpStr.find(" ");
            sign = tmpStr.substr(0, tmp_2);
            ////expression 2
            exp_2 = tmpStr.substr(tmp_2 + 1, tmpStr.size() - tmp_2 - 2);
            if (exp_2.find("NOT") != string::npos)
            {
                exp_2 = notFunction(exp_2);
            }

            //recursive on expression 1
            a = expression(exp_1, 0);
            // recursive on expression 2
            b = expression(exp_2, 0);
```

```
            // the  we return the outcome
            if (sign == "AND")
                return (a && b);
            else
                return (a || b);
        }
    }
    //this returns the negation of a logic.
    string notFunction(string str)
    {
        int tmp = str.find("NOT");
        string tmp_expression = str.substr(tmp + 3, str.size());
        int tmp_closing = tmp_expression.find(")");
        tmp_expression = tmp_expression.substr(0, tmp_closing);
        bool before;

        if (tmp_expression.find(" ") != string::npos)
        {
            before = expression(tmp_expression, 0);
        }
        else
        {
            tmp_expression = tmp_expression.substr(1, tmp_closing - 1);
            before = expression(tmp_expression, 0);
        }

        string notExp = before == true ? "NOT(TRUE)" : "NOT(FALSE)";
        return notExp;
    }
    void calculate(string input)
    {
        if (checkValidity(input) && !input.empty())
        {
            if (expression(input, 0) == true)
                cout << "TRUE" << endl;
            else
                cout << "FALSE" << endl;
        }
        else
            cout << "\t ERROR!!!" << endl;
    }
```

```
int main()
{
    //start
    ProceedInput();

    return 0;
}
```