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 10

Студент гр. 8304 _____ Чешуин Д.И.

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.

Вариант 10: Построить синтаксический анализатор для определяемого далее по-нятия константное_выражение.
константное_выражение::=ряд_цифр|
константное_выражение знак_операции константное_выражение
знак_операции::=+ | - | *
ряд_цифр::=цифра | цифра ряд_цифр

Описание алгоритма.

Для вычисления значения выражения методом рекурсии необходимо считать строку, затем рекурсивно проверить корректность выражения.
При проверке на соответствие определению «константное_выражение», проверяется либо наличие конструкции «ряд_цифр», либо наличие конструкции «ряд_цифр знак_операции константное_выражение». Конструкция «ряд_цифр» проверяется на наличие цифры, либо конструкцию «цифра ряд_цифр». Алгоритм возвращает false если строка не соответствует определению, и true в ином случае. После завершение алгоритма итератор должен находиться в конце строки, в противном случае строка некорректна.

```
проверить_выражение(позиция)
        если проверить_ряд_цифр(позиция) = true
                позиция++
```

```
если конец_строки(позиция)
        вернуть True
иначе

если проверить_знак(позиция) = true
        позиция ++
иначе
        вернуть false
если проверить_выражение(позиция) = true
        позиция += 1
иначе
        вернуть false
```

Спецификация программы.

Программа предназначена для синтаксического анализа выражения методом рекурсии.

Программа написана на языке C++. Входными данными является строка. Выходными данными являются промежуточные значения проверки выражения и конечный результат (корректна строка или нет).

Тестирование.

Ввод	Вывод
0 1 2 3 4 5 6 7 8 9	OK

1 + 2 - 3 * 4	OK
10	Failed
+	Failed
1 ++ 2	Failed

Анализ алгоритма.

Для экономии памяти передается ссылка на строку, что позволяет избежать копирования. Алгоритм работает за линейное время. Недостаток рекурсивного алгоритма – ограниченный стек вызовов функций, что в свою очередь накладывает ограничение на размер входной строки, а также снижается производительность за счёт затрат на большое количество вызова функций.

Описание функций и СД

Функции, реализующие интерфейс синтаксического анализатора.

```
bool parse(types type, string& str);
```

Функция, которая принимает на вход ожидаемый тип из перечисления и ссылку на проверяемую строку. Осуществляет синтаксический анализ переданной строки на совпадение с ожидаемым типом. Возвращает true, если тип строки совпадает с ожидаемым, и false – если не совпадает

bool typeIsEq(char chr, types type);

Функция принимает на вход терминальный символ и ожидаемый тип символа. Возвращает true, если тип символа совпадает с ожидаемым, и false – если не совпадает

Выводы.

В ходе работы я познакомился с рекурсией, научился работать с данным методом решения задач. Я считаю, что метод рекурсии для данной задачи проверки выражения на корректность не эффективен.

Приложение А.

main.cpp

```
#include <iostream>
#include <fstream>
#include <string>
#include <cstring>

using namespace std;

enum types{
    CONST_EXPR,
    SERIES_OF_NUMBER,
    OPERATION_SIGN,
    DIGIT,
    SPACE,
};

int dialog();
void cliHandler(int argc, char** argv);
int readCommand(void);
bool parse(types type, string& str);
bool typeIsEq(char chr, types type);
string typeToStr(types type);

int main(int argc, char** argv) {

    if(argc > 1){
        cliHandler(argc, argv);
    }
    else{
        dialog();
    }

    return 0;
}

bool parse(types type, string& str){
    static unsigned int position = 0;

    static int deep = 0;
    static int instanceCount = 0;
    int instanceNum;

    if(deep == 0){
        position = 0;
        instanceCount = 0;
    }

    deep += 1;
    instanceCount += 1;

    instanceNum = instanceCount;

    cout << "Instance " << instanceNum << " Deep - "
         << deep << " Expected type - " << typeToStr(type) << endl;

    if(position == str.length()){
        cout << "Instance " << instanceNum << " Deep - " << deep << " Expected type - "
             << typeToStr(type) << " failed - end of string" << endl;
        return false;
    }

    int isCorrect = false;

    switch(type){
```

```
        case CONST_EXPR:{
            if(parse(SERIES_OF_NUMBER, str)){
                if(position + 1 >= str.length()){
                    isCorrect = true;
                    break;
                }
                position += 1;

                if(parse(OPERATION_SIGN, str)){
                    isCorrect = parse(CONST_EXPR, str);
                }
            }
            break;
        }

        case SERIES_OF_NUMBER:{
            if(typeIsEq(str[position], DIGIT)){
                if(position + 1 == str.length()){
                    isCorrect = true;
                    break;
                }
                if(typeIsEq(str[position + 1], SPACE)){
                    if(typeIsEq(str[position + 2], DIGIT)){
                        position += 2;
                        isCorrect = parse(SERIES_OF_NUMBER, str);
                    }
                    else{
                        isCorrect = true;
                    }
                }
            }
            break;
        }

        case OPERATION_SIGN:{
            if(typeIsEq(str[position], SPACE)){
                if(typeIsEq(str[position + 1], OPERATION_SIGN)){
                    if(typeIsEq(str[position + 2], SPACE)){
                        position += 3;
                        isCorrect = true;
                    }
                }
            }
            break;
        }

        default:{
            deep -= 1;

            cout << "Instance " << instanceNum << " Deep - " << deep << " Expected type - "
                 << typeToStr(type) << endl << " failed - incorrect expected type" << endl;

            return false;
        }
    }

    cout << "Instance " << instanceNum << " Deep - " << deep << " Expected type - "
         << typeToStr(type);

    if(isCorrect){
        cout << " - OK" << endl;
    }
    else{
        cout << " - failed - real type not equal with expected" << endl;
    }

    deep -= 1;

    return isCorrect;
}

bool typeIsEq(char chr, types type){
    if(chr == ' ' && type == SPACE){
        return true;
    }
    else if(isdigit(chr) && type == DIGIT){
        return true;
    }
    else if((chr == '+' || chr == '-' || chr == '*') && type == OPERATION_SIGN){
```

```
        return true;
    }
    else{
        return false;
    }
}

string typeToStr(types type){
    switch (type) {
        case CONST_EXPR:
            return "Const expression";

        case SERIES_OF_NUMBER:
            return "Series of number";

        case OPERATION_SIGN:
            return "Operation sign";

        case DIGIT:
            return "Number";

        case SPACE:
            return "Space";

    }

    return "Unknown type";
}

int dialog(void){
    int command;
    string expression;
    string filePath;
    ifstream inputFile;
    ofstream resultFile;

    while(true){
        cout << "1. Validate expression from console." << endl;
        cout << "2. Validate all expressions from file." << endl;
        cout << "3. Exit." << endl;
        cout << "Enter command." << endl;

        command = readCommand();

        switch(command){
            case 1:
                cout << "Enter expression." << endl;

                cin.clear();
                getline(cin, expression);

                cout << endl << "Expression to validate: " << expression << endl;
                if(parse(CONST_EXPR, expression) == 0){
                    cout << "Result - not const expression." << endl << endl;
                }
                else{
                    cout << "Result - const expression." << endl << endl;
                }
            break;

            case 2:
                cout << "Enter path of file." << endl;

                cin.clear();
                getline(cin, filePath);

                inputFile.open(filePath);

                if(inputFile.is_open()){
                    resultFile.open(filePath + " - result");

                    cout << "Opened file - " << filePath << endl;

                    while(getline(inputFile, expression)){
                        cout << endl << "Expression to validate: " << expression << endl;
                        resultFile << expression;

                        if(parse(CONST_EXPR, expression)){
                            cout << "Result - const expression." << endl << endl;
```

```
                            resultFile << " - OK" << endl;
                        }
                        else{
                            cout << "Result - not const expression." << endl << endl;
                            resultFile << " - FAILED" << endl;
                        }
                    }

                    inputFile.close();
                    resultFile.close();
                }
                else{
                    cout << "Cant't open file - " << filePath << ". Try again." << endl << endl;
                }
            break;

            case 3:
                return 0;
        }
    }
}

int readCommand(void){
    string command;

    while(true){
        cin.clear();
        getline(cin, command);

        if(command == "1"){
            return 1;
        }
        else if (command == "2") {
            return 2;
        }
        else if (command == "3") {
            return 3;
        }
        else{
            cout << command << " - is incorrect command. Try again." << endl << endl;
        }
    }

}

void cliHandler(int argc, char** argv){
    string expression;
    string filePath;
    ifstream inputFile;
    ofstream resultFile;

    for(int i = 1; i < argc; i++){
        if(strcmp(argv[i], "--file") == 0){
            i += 1;

            filePath = argv[i];

            inputFile.open(filePath);

            if(inputFile.is_open()){
                resultFile.open(filePath + " - result");

                cout << "Opened file - " << filePath << endl;

                while(getline(inputFile, expression)){
                    cout << endl << "Expression to validate: " << expression << endl;
                    resultFile << expression;

                    if(parse(CONST_EXPR, expression)){
                        cout << "Result - const expression." << endl << endl;
                        resultFile << " - OK" << endl;
                    }
                    else{
                        cout << "Result - not const expression." << endl << endl;
                        resultFile << " - FAILED" << endl;
                    }
                }

                inputFile.close();
```

```
                resultFile.close();
            }
            else{
                cout << "Cant't open file - " << filePath << ". Try again." << endl << endl;
            }
        }
    }
}
```