МИНОБРНАУКИ РОССИИ

САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ

ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)

Кафедра МО ЭВМ

ОТЧЁТ

по лабораторной работе №1

по дисциплине «Алгоритмы и структуры данных»

Тема: Рекурсия

Студент гр. 8381 _____ Королёва Д.С.

Преподаватель _____ Жангиров Т.Р.

Санкт-Петербург

2019

Цель работы

Ознакомиться с основными понятиями и приёмами рекурсивного программирования, получить навыки программирования рекурсивных процедур и функций на языке программирования C++. Разработать программу, использующую рекурсию, и сопоставить рекурсивное решение с итеративным решением задачи.

Задание

10. Построить синтаксический анализатор для определяемого далее понятия константное_выражение.

константное_выражение ::= ряд_цифр | константное_выражение знак_операции константное_выражение

знак_операции ::= + | - | *

ряд_цифр ::= цифра | цифра ряд_цифр

Основные теоретические положения

Синтаксический анализатор (парсер) - программа, которая определяет, является ли заданная (входная) последовательность символов *константным_выражением* или нет. В случае ответа «нет» сообщается место и причина ошибки. Синтаксический анализатор удобно реализовать с помощью рекурсии. Функция, проверяющая выражение, может запускать внутри себя функции, проверяющие отдельные части выражения на соответствие заданным паттернам.

Выполнение работы

Написание работы производилось на базе операционной системы Windows 8 в среде разработки Visual Studio Code. Сборка, отладка и тестирование также производились в Visual Studio Code с использованием компилятора MinGW.

Для реализации программы был разработан CLI с несколькими вариантами ввода (через аргументы командной строки, с консоли, из файла).

Основные функции алгоритма ConstExpression(), OperationSign() и NumberSequence() определяют структуру константного выражения. Эти функции проверяют начало полученной строки на соответствие следующим условиям: *константное_выражение, знак операции, ряд цифр*. Все эти функции возвращают булевские значения, который показывает, соответствует ли начало переданной строки str паттерну функции. Аргументы:

string& str – обрабатываемая строка;

int& pos – номер текущего символа относительно начала исходной строки.

Вспомогательные функции: NextSymbol() сдвигает строку на 1 символ, ErrorOut() выводит сообщение об ошибке.

В главной программе происходит ввод данных (функция ProceedInput()), вызов функции ConstExpression() и вывод результата "OK" или "ERROR" в зависимости от исходного выражения.

Оценка сложности алгоритма

Алгоритм, реализованный в программе, имеет линейную зависимость от длины строки, то есть сложность оценивается как O(n). Рассуждения отталкиваются от того, что в алгоритме отсутствуют циклы по одному

символу, а на каждом шаге рекурсии из строки удаляется как минимум один символ. Ввиду рекурсивного алгоритма рост занимаемой памяти также растет линейно из-за создаваемых в функциях временных переменных. Для снижения потребления памяти объёмные переменные в большинстве функций передаются по ссылке.

Тестирование программы

Таблица 1 — Тестирование программы

Входные данные	Результат работы
5+6*77	Lab work #1 Choose your input 0 - from console 1 - from file 0 Your input: 5+6*77 5+6*77 OK
5+6*7**	Lab work #1 Choose your input 0 - from console 1 - from file 1 Filename: input.txt 5+6*7* Number sequence expected ERROR

Выводы

В ходе выполнения лабораторной работы был изучен такой вид алгоритмов, как синтаксические анализаторы. Была реализована программа, которая анализирует строку рекурсивным методом, определяя соответствие определению.

Приложение А

Исходный код программы

Название файла: main.cpp

```
#include <iostream>
#include <fstream>
#include <string>
#include <algorithm>
#include <memory>
#include <cmath>
#include <cctype>
#include <cstring>
using namespace std;

string ProceedInput(int argc, char** argv);
string ReadFromFile(std::string filename);
string ReadFromConsole();
void ErrorOut(const string& str);
void NextSymbol(string & str, int & pos);
bool ConstExpression(string & str, int & pos);
bool OperationSign(string & str, int & pos);
bool NumberSequence(string & str, int & pos, bool found = fa
lse);

int main (int argc, char** argv) {
    string input = ProceedInput(argc, argv);
    int pos = 0;
```

```
    bool result = ConstExpression(input, pos);
    if (result) cout << endl << "OK" << endl;
    else cout << endl << "ERROR" << endl;
    return 0;
}

string ProceedInput (int argc, char** argv) {
    cout << "Lab work #1" << endl;
    if (argc > 1) {
        cout << "Reading from argv..." << endl;
        string ins(argv[1]);
        return ins;
    }
    else {
        cout << "Choose your input" << endl;
        cout << "0 - from console" << endl;
        cout << "1 - from file" << endl;
        int num = 0;
        cin >> num;
        string input;
        switch (num) {
            case 0:
                cout << "Your input: ";
                return ReadFromConsole();
            case 1:
```

```
                cout << "Filename: ";
                cin >> input;
                return ReadFromFile(input);
            default:
                return ProceedInput(0, nullptr);
        }
    }
}

string ReadFromFile (std::string filename) {
    ifstream infile(filename);
    if (!infile) {
        cout << "File can't be open!" << endl;
        return "";
    }
    string res;
    infile >> res;
    return res;
}

string ReadFromConsole () {
    string res;
    cin >> res;
    return res;
}
```

```
void NextSymbol (string& str, int& pos) {
    cout << str[0];
    str = str.substr(1);
    pos++;
}

void ErrorOut (const string& str) {
    static bool first = true;
    if (first) {
        cout << endl << str;
        first = false;
    }
}

//константное_выражение::= ряд_цифр|константное_выражение зн
ак_операции константное_выражение
bool ConstExpression (string& str, int& pos) {
    if (str.length() == 0)
        return false;
    if (NumberSequence(str, pos)) {
        if (OperationSign(str, pos)) {
            if (ConstExpression(str, pos)) {
                return true;
            }
            else {
```

```
                ErrorOut("Const expression expected");
                return false;
            }
        }
        return true;
    }
    ErrorOut("CNumber sequence expected");
    return false;
}

//знак_операции::= + | - | *
bool OperationSign (string& str, int& pos) {
    if (str.length() == 0)
        return false;
    if (str.length() > 0 && (str[0] == '+' || str[0] == '-
' || str[0] == '*')) {
        NextSymbol(str, pos);
        return true;
    }
    ErrorOut("Sign expected");
    return false;
}

//ряд_цифр::= цифра | цифра ряд_цифр
bool NumberSequence (string& str, int& pos, bool found) {
    if (str.length() == 0)
```

```
        return false;
    if (str.length() > 0 && isdigit(str[0])) {
        NextSymbol(str, pos);
        if (NumberSequence(str, pos, true)) {
            return true;
        }
        return true;
    }
    if (!found) ErrorOut("Number sequence expected");
    return false;
}
```