

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

Студент гр. 8381

Сахаров В.М.

Преподаватель

Жангиров Т.Р.

Санкт-Петербург

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.

Построить синтаксический анализатор для определённого далее понятия `логическое_выражение`.

`логическое_выражение ::= TRUE | FALSE | идентификатор | NOT (операнд) | операция (операнды)`

`идентификатор ::= буква`

`операция ::= AND | OR`

`операнды ::= операнд | операнд, операнды`

`операнд ::= логическое_выражение`

Основные теоретические положения.

Синтаксический анализатор (парсер) - программа, которая определяет, является ли заданная (входная) последовательность символов *логическим_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 производились в JetBrains CLion с использованием компилятора MinGW.

Для реализации программы был разработан CLI с несколькими вариантами ввода (с консоли, из файла, случайная строка из массива примеров)

После получения входной строки запускается вспомогательная функция инициализации проверки `CheckStatement()`, которая проверяет, что после выполнения основного алгоритма в строке не осталось лишних символов, а также показывает, на каком из символов произошла ошибка.

Основные функции алгоритма `Statement()`, `Name()`, `Operation()`, функции возвращают `bool`, который показывает, соответствует ли начало переданной строки `str` паттерну функции. Аргументы `pos` и `indent` требуются для вывода ошибочного символа и прогресса работы алгоритма.

Вспомогательные функции: `FindWord()` выполняет поиск строки `aim` в начале заданной, а также проверкой того, что следующий символ отделён от найденной любым терминальным символом. `Skip()` сдвигает строку на заданное количество символов, по умолчанию 1. Требуется для удаления найденных слов в `FindWord()`. `SkipSpaces()` опциональна, так как в задании не сказано чётко - игнорируются ли пробелы алгоритмом. Данная функция удаляет все пробелы в начале строки после удаления символов функцией `Skip()`.

После завершения работы функции `CheckStatement()` производится вывод строк `SUCCESS` или `ERROR` в зависимости от результата алгоритма

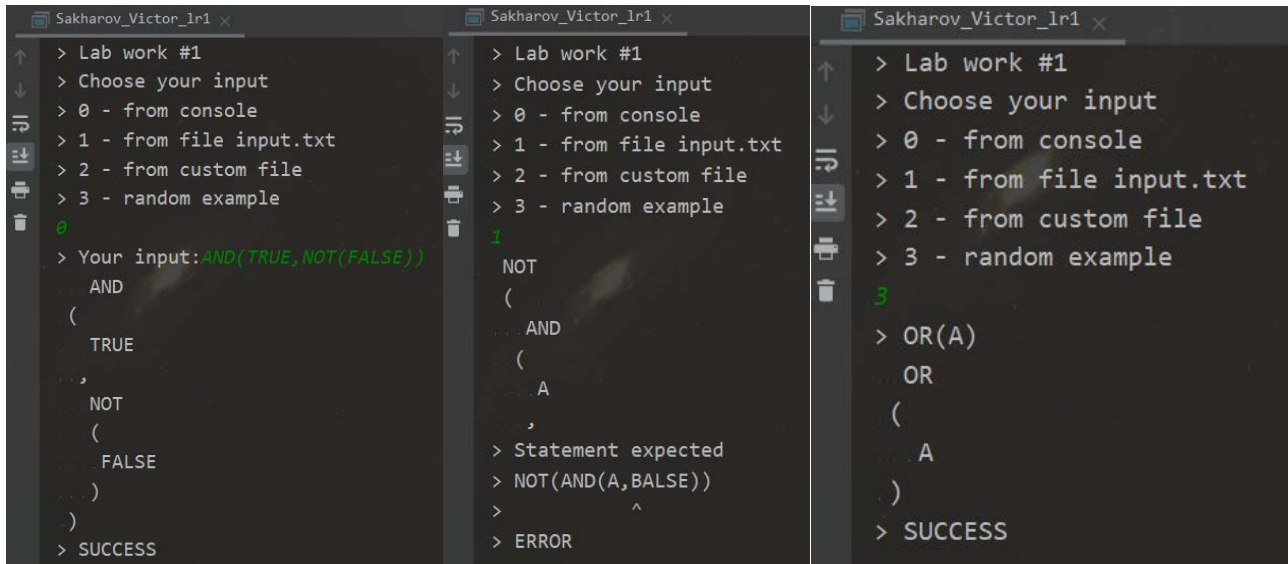
Оценка эффективности алгоритма.

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 растёт линейно из-за создаваемых в функциях временных переменных. Для снижения потребления памяти объёмные переменные в большинстве функций передаются по ссылке.

Тестирование программы.

Пример выходных данных:



```
> Lab work #1
> Choose your input
> 0 - from console
> 1 - from file input.txt
> 2 - from custom file
> 3 - random example
0
> Your input: AND(TRUE, NOT(FALSE))
AND
(
  TRUE
,
  NOT
  (
    FALSE
  )
)
> SUCCESS
```

```
> Lab work #1
> Choose your input
> 0 - from console
> 1 - from file input.txt
> 2 - from custom file
> 3 - random example
1
NOT
(
  AND
  (
    A
  ,
  > Statement expected
  > NOT(AND(A, BALSE))
  > ^
  > ERROR
```

```
> Lab work #1
> Choose your input
> 0 - from console
> 1 - from file input.txt
> 2 - from custom file
> 3 - random example
3
> OR(A)
OR
(
  A
)
> SUCCESS
```

Выводы.

В ходе выполнения лабораторной работы был изучен такой вид программ, как синтаксические анализаторы. Была реализована программа, которая анализирует строку рекурсивным методом, определяя соответствие определению. Было выявлено, что для части рекурсивных определений можно создать универсальные алгоритмы (что, например, реализовано в программе JetBrains MPS для создания языков программирования).

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.cpp

```
#include "LabExec.h"
#include "LabIo.h"

int main (int argc, char* argv[]) {
    string input = ProceedInput(argc, argv);
    if (input.empty()) return 0;
    bool success = CheckStatement(input);
    cout << "> " << (success ? "SUCCESS" : "ERROR") << endl;
    return 0;
}
```

Название файла: structs.h

```
#ifndef STRUCTS_H
#define STRUCTS_H
#include <iostream>
#include <fstream>
#include <string>
#include <algorithm>
#include <memory>
#include <random>
#include <cmath>
#include <cctype>
#include <cstring>
#include <ctime>
using namespace std;
#endif //STRUCTS_H
```

Название файла: LabIo.cpp

```
#include "LabIo.h"

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 input.txt" << endl;
        cout << "> 2 - from custom file" << endl;
        cout << "> 3 - random example" << endl;
        int command = 0;
        cin >> command;
        string input;

        switch (command) {
            case 0:
                cout << "> Your input: ";
                return ReadFromConsole();
            case 1:
                return ReadFromFile("input.txt");
            case 2:
                cout << "> Filename: ";
                cin >> input;
                return ReadFromFile(input);
            case 3:
                return ReadRandomFromExamples();
            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;
}

string ReadRandomFromExamples () {
    std::mt19937 mt(time(nullptr));
    std::uniform_int_distribution<int> dist(0, 9);
    std::string res = string(Examples[dist(mt)]);
    cout << "> " << res << endl;
    return res;
}

void ProceedOutput (const std::string output, const int indent)
{
    for (int i = 0; i < indent; ++i) {
        cout << " ";
    }
    cout << output << endl;
}

void ProceedError (const std::string& error, int& pos) {
    if (pos != -1) {
        pos = -1;
        cout << "> " << error << endl;
    }
}

```

```
}
```

Название файла: LabExec.cpp

```
#include "LabExec.h"
```

```
void SkipSpaces (std::string& str, int& pos) {  
    while (str.length() > 0 && str[0] == ' ') {  
        str = str.substr(1);  
        pos++;  
    }  
}
```

```
void Skip (std::string& str, int& pos, const int indent, int  
n) {  
    if (str.length() >= n) {  
        ProceedOutput(str.substr(0, n), indent);  
        str = str.substr(n);  
        pos++;  
        SkipSpaces(str, pos);  
    }  
}
```

//Проверка, что str начинается с aim и дальше находится
не_буква

```
bool FindWord (std::string& str, int& pos, const int indent,  
const char* aim) {  
    int len = strlen(aim);  
    if (!str.compare(0, len, aim) && (len == str.length() ||  
(len < str.length() && (!isalnum(str[len])  
|| !isalnum(aim[0]))))) {  
        Skip(str, pos, indent, len);  
        return true;  
    }  
    return false;  
}
```



```

bool CheckStatement (std::string& str) {
    std::string copy = string(str);
    int position = 0;
    if (Statement(copy, position, 1)) {
        if (copy.empty()) {
            return true;
        }
        else ProceedError("End of string expected. \"" + copy
+ "\" left", position);
    }
    cout << "> " << str << endl << "> ";
    for (int i = 0; i < str.length() - copy.length(); ++i) {
        cout << " ";
    }
    cout << "^" << endl;
    return false;
}

```

```

// TRUE      |   FALSE   | Name   | NOT (Operand) | Operation
(OperandList)

```

```

bool Statement (std::string& str, int& pos, const int indent)
{
    if (FindWord(str, pos, indent, TRUE_S)) {
        return true;
    }
    else if (FindWord(str, pos, indent, FALSE_S)) {
        return true;
    }
    else if (Name(str, pos, indent)) {
        return true;
    }
    else if (FindWord(str, pos, indent, NOT_S)) {
        if (FindWord(str, pos, indent, OPEN_BRACKET)) {

```

```

        if (Operand(str, pos, indent + 1)) {
            if (FindWord(str, pos, indent, CLOSE_BRACKET))
{
                return true;
            }
            else ProceedError("'')' expected", pos);
        }
        else ProceedError("Operand expected", pos);
    }
    else ProceedError("'(' expected", pos);
}

else if (Operation(str, pos, indent + 1)) {
    if (FindWord(str, pos, indent, OPEN_BRACKET)) {
        if (OperandList(str, pos, indent + 1)) {
            if (FindWord(str, pos, indent, CLOSE_BRACKET))
{
                return true;
            }
            else ProceedError("'')' expected", pos);
        }
        else ProceedError("List of operands expected",
pos);
    }
    else ProceedError("'(' expected", pos);
}
ProceedError("Statement expected", pos);
return false;
}

// Letter
bool Name (std::string& str, int& pos, const int indent) {
    if ((str.length() == 1 && isalpha(str[0])) ||
(str.length() > 1 && isalpha(str[0]) && !isalnum(str[1]))) {
        Skip(str, pos, indent);
    }
}

```

```

        return true;
    }
    return false;
}

// AND | OR
bool Operation (std::string& str, int& pos, const int indent)
{
    if (FindWord(str, pos, indent, AND_S)) {
        return true;
    }
    else if (FindWord(str, pos, indent, OR_S)) {
        return true;
    }
    return false;
}

// Operand | Operand, OperandList
bool OperandList (std::string& str, int& pos, int indent, bool
first) {
    if (Operand(str, pos, indent + 1)) {
        if (FindWord(str, pos, indent, COMMA)) {
            return OperandList(str, pos, indent, false);
        }
        return true;
    }
    //if (first) ProceedError("Operand expected", pos);
    return false;
}

// Statement
bool Operand (std::string& str, int& pos, const int indent) {
    return Statement(str, pos, indent);
}

```

