

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

ОТЧЕТ
по лабораторной работе №1
по дисциплине «Алгоритмы и структуры данных»
Тема: Рекурсия

Студент гр. 9304

Афанасьев А.

Преподаватель

Филатов А. Ю.

Санкт-Петербург

2020

Цель работы.

Научиться применять рекурсию в программировании.

Постановка задачи.

Вариант 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

Также дополнительно была внедрена поддержка скобок.

Выполнение работы:

Программа на вход принимает путь до файла, в котором лежит строка для анализа. Реализован класс `Analyzer`, который занимается проверкой поданной строки. Также есть 10 тестов разных видов для тестирования алгоритма. Саму же проверку можно вызвать методом `analyzeCnstExpr()`, предварительно сконструировав объект. Перечисление `Status` сделано для идентификации объектов в строке, чтобы следить за комбинациями этих объектов.

`analyzeCnstExpr()` работает рекурсивно, он умеет различает несколько объектов: ряд цифр, знаки операций и скобки. Определив, что встретилось в строке, алгоритм начинает выполнять те или иные действия в зависимости от того, что встретилось. Например, встретив, ряд цифр, алгоритм проверит, был ли знак в прошлом вызове, если нет и это не начало строки, то вернет логический ноль, прервав тем самым рекурсию. А, если знак был или это начало строки, то он вызовет себя уже после этого ряда цифр, запомнив, что последним встреченным объектом был ряд цифр. А, если встретит открывающую скобку после знака или начала строки, то вызовет самого себя

уже в границах этих скобок, если, конечно, он проверил наличие закрывающей скобки, иначе вернется логический нуль. Еще алгоритм возвращает нуль, если строка закончилась на знаке или была пуста вовсе, для всего этого прописаны условные операторы.

Пробелы и знаки табуляции игнорируются, алгоритм их просто перешагивает.

Приватный метод `skipSome()` всего лишь пропускает встреченную последовательность цифр или пробелов, перемещая курсор на следующий символ после нее. Создан ради отсутствия дублирования кода.

Также было посчитано, что скорость алгоритма составляет $O(n)$, так как он проходит вдоль строки один раз.

Тестирование.

Программа собирается через Makefile командой `make`, после чего создается файл `program.out`. Программе при запуске можно передавать название файла с одним тестом, а 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. На выход возвращается 1 или 0, если вы подаете тесты самостоятельно, или `success` или `fail`, если запускаете скрипт.

Таблица 1 – Тесты и их результаты

№	Входные данные	Выход
0	+	0
1	+2+2	1
2		0
3	---***+	0
4	$(213213 + 213213213) + (-232) * (((2323) + 2))$	1
5	$324 + (123123 + 6546) -$	0
6	$232+++32$	0
7	$324324 + (324324 + 3243242345 + 23$	0
8	$*2323* 2323$	0
9	$-((23423432) + 324324) + 123456 + ((23) + (223432432 + (2 + (2 + 2)))) + 2) + (1 * 232) * ((2323 + 2323) + (23213 - 2323)) + (20) + 2$	1

Комментарии к некоторым тестам.

1. После знака операции “+” не было встречено константное выражение.
5. После знака операции “-” не было встречено константное выражение. Строка не может заканчиваться знаком.
6. После знака операции встречен знак операции.
7. Не найдена соответствующая закрывающая скобка.
8. Строка не может начинаться со знака “ * ”.

Вывод.

В ходе выполнения работы ознакомились с основными понятиями и приемами рекурсивного программирования и научились ее применять. Также реализовали поддерживающий скобки синтаксический анализатор, который

работает рекурсивно за $O(n)$. В приложении А можно ознакомиться с исходным кодом.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД

main.h.

```
#ifndef MAIN_H
#define MAIN_H

#include "Analyzer.h"
#include <iostream>
#include <fstream>

#endif
```

main.cpp.

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

int main(int amount, char **args)
{
    std::string strToCheck;

    std::ifstream file;
    file.open(args[1], std::ios_base::in);

    std::getline(file, strToCheck);

    file.close();

    Analyzer analyzer(strToCheck);
    std::cout << analyzer() << '\n';

    return 0;
}
```

Analyzer.h.

```
#ifndef ANALYZER_H
#define ANALYZER_H

#include <string>

enum Status
```

```

{
    kStart,
    kOper,
    kNumber,
    kNothing
};

class Analyzer
{
    const std::string &str;
    size_t index;
    size_t lastIndex;
    Status oldObjStatus;
    inline void skipSome(const char *thing);

public:
    const bool operator()();
    Analyzer(const std::string &otherStr, size_t start
= 0, size_t end = std::string::npos);
    ~Analyzer() = default;
};

#endif

```

Analyzer.cpp.

```

#include "../libs/Analyzer.h"

Analyzer::Analyzer(const std::string &otherStr, size_t
start, size_t end) : str(otherStr), index(start)
{
    lastIndex = (end == std::string::npos) ?
otherStr.length() : end;
    oldObjStatus = kStart;
}

const bool Analyzer::operator()()
{
    Status currentObj = kNothing;
    if (index < lastIndex)
    {
        if (str[index] == '-' || str[index] ==
'+' || str[index] == '*')

```

```

{
    if (oldObjStatus == kOper ||
        (str[index] == '*' && oldObjStatus
        == kStart))
        return 0;
    currentObj = kOper;
    ++index;
}
else if (str[index] >= '0' && str[index]
<= '9')
{
    if (oldObjStatus == kNumber)
        return 0;
    currentObj = kNumber;
    skipSome("1234567890");
}
else if (str[index] == '(')
{
    if (oldObjStatus != kOper &&
        oldObjStatus != kStart)
        return 0;
    long long bar = 1;
    size_t firstIndex = index + 1;
    do
    {
        ++index;
        if (str[index] == ')')
            --bar;
        else if (str[index] == '(')
            ++bar;
    } while (bar != 0 && index <
        lastIndex);
    Analyzer analyzer(str, firstIndex,
        index);
    if (bar != 0 || analyzer() == 0)
        return 0;
    ++index;
    currentObj = kNumber;
}
else if (str[index] != ' ' &&
    str[index] != '\t')
    return 0;

```



```

        if (str[index] == ' ' || str[index] == '\t')
            skipSome(" \t");
    }
    if ((oldObjStatus == kOper || oldObjStatus == kStart) && currentObj == kNothing)
        return 0;
    else if (currentObj == kNothing && oldObjStatus != kNothing)
        return 1;
    oldObjStatus = currentObj;
    return ((*this)()) ? 1 : 0;
}

inline void Analyzer::skipSome(const char *thing)
{
    index = str.find_first_not_of(thing, index + 1);
    if (index > lastIndex)
        index = lastIndex;
}

```

Makefile.

```

compiler = g++
flags = -c -g
appname = lab1
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)

```