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 _____ Фиалковский М. С.

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 Mode сделано для того, что чтобы ввести

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 нуль, 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.

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

Тестирование.

Программа собирается через Makefile командой make, после чего создатся 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

№ теста	Входные данные	Результат
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	1
8	*2323* 2323	0
9	-((23423432)+ 324324)+123456 + ((23)+(223432432+(2+(2+ 2)))+ 2) +(1 * 232) *((2323+2323) + (23213 -2323)) + (20) + 2	1

Вывод.

В ходе выполнения работы научились применять рекурсию в программировании и написали синтаксический анализатор.

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.analyzeCnstExpr() << '\n';

    return 0;
}
```

Analyzer.h.

```
#ifndef ANALYZER_H
#define ANALYZER_H

#include <string>

enum Mode
```

```
{
    kStart = 0,
    kOper = 1,
    kNumber = 2,
    kNothing = -1
};

struct Object
{
    Mode mode;
    char ch;
    Object() : mode(kStart), ch(0) {}
};

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

public:
    const bool analyzeCnstExpr();
    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;
}
```

```
const bool Analyzer::analyzeCnstExpr()
{
     Object modeNew;
     modeNew.mode = kNothing;
     if (index < lastIndex)
     {
          if (str[index] == '-' || str[index] == '+' ||
          str[index] == '*')
          {
               if (!(oldMode.mode == kOper ||
               (str[index] == '*' && oldMode.mode
               == kStart)))
               {
                    modeNew.mode = kOper;
                    modeNew.ch = str[index];
                    ++index;
               }
               else
                    return 0;
          }
          else if (str[index] >= '0' && str[index]
          <= '9')
          {
               if (oldMode.mode != kNumber)
               {
                    modeNew.mode = kNumber;
                    skipSome("1234567890");
               }
               else
                    return 0;
          }
          else if (str[index] == '(')
          {
               if (oldMode.mode == kOper ||
               oldMode.mode == kStart)
               {
                    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.analyzeCnstExpr() == 1)
                {
                    ++index;
                    modeNew.mode = kNumber;
                }
                else
                    return 0;
            }
            else
                return 0;
        }
        else if (str[index] != ' ' &&
        str[index] != '\t')
            return 0;

        if (str[index] == ' ' || str[index] == '\t')
            skipSome(" \t");
    }
    if ((oldMode.mode == kOper || oldMode.mode ==
    kStart) && modeNew.mode == kNothing)
        return 0;
    else if (modeNew.mode == -1 && oldMode.mode >= 0)
        return 1;
    oldMode = modeNew;
    return (analyzeCnstExpr()) ? 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 = program.out
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)
```