ММИНОБРНАУКИ 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: Взаимно-рекурсивные функции и процедуры. Синтаксический анализатор понятия скобки.

Студент гр. 9384 _____ Николаев А.А.

Преподаватель _____ Ефремов М.А.

Санкт-Петербург

2020

Цель работы.

Реализовать синтаксический анализатор понятия скобки на основе взаимной рекурсии.

Задание.

ВАРИАНТ 12.

12. Построить синтаксический анализатор для понятия скобки.

скобки::=квадратные | круглые | фигурные
квадратные::=[круглые фигурные] | +
круглые::=(фигурные квадратные) | -
фигурные::={квадратные круглые} | 0

Выполнение работы.

В начале программа запрашивает у пользователя путь до файла из которого требуется считать данные и отправить их на обработку и название файла, в который будет записан результат работы программы. Программа построчно считывает данные из входного файла и обрабатывает их. В самом начеле обработки строки вызывается функция:

bool brackets(std::string expression, int& index)

которая берет первый символ из строки и проверяет, что он является одной из нужных нам скобок. Если он определил данный символ, как один из удовлетворяющих нашему условию, то он вызывает соответствующую функцию для обработки того, что находится внутри данной скобки, предварительно сместив указатель, на следующий элемент строки.

. Функция

bool isSquareBrackets (std::string expression, int& index)

проверяет, что бы внутри квадратной скобки находились круглые и фигурные или символ +. В первом случае, когда она встретит скобки, она вызовет следующую функцию для обработки полученого вида скобок.

bool isRoundBrackets (std::string expression, int& index)

проверяет, что бы внутри круглой скобки находились фигурные и квадратные или символ -. В первом случае, когда она встретит скобки, она вызовет следующую функцию для обработки полученого вида скобок.

bool isFiguredBrackets (std::string expression, int& index)

проверяет, что бы внутри фигурной скобки находились квадратные и круглые или символ 0. В первом случае, когда она встретит скобки, она вызовет следующую функцию для обработки полученого вида скобок.

Тестирование.

Результаты тестирования представлены в табл. 1.

Таблица 1 – Результаты тестирования

Входные данные	Выходные данные
[+]	[+] Yes
(-)	(-) Yes
{0}	{0} Yes
[(-){0}]	[(-){0}] Yes
({0}[(-){[+](-)}])	({0}[(-){[+](-)}]) Yes
[]	[] No
{[][]}	{[][]} No
{	{ No
abc	abc No
[}	[} No
{}[}}	{}[}} No

Выводы.

В ходе выполнения данной лабораторной работы были получены навыки по работе с рекурсивными функциями, а также, разработан синтаксический анализатор понятия скобки.

ПРИЛОЖЕНИЕ
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.cpp

```
/*
12. Построить синтаксический анализатор для понятия скобки.
скобки::=квадратные | круглые | фигурные
квадратные::=[круглые фигурные] | +
круглые::=(фигурные квадратные) | -
фигурные::={квадратные круглые} | 0
*/

#include <iostream>
#include <fstream>
#include <string>
#include <iomanip>

bool brackets(std::string expression, int& index);
bool isSquareBrackets(std::string expression, int& index);
bool isRoundBrackets(std::string expression, int& index);
bool isFiguredBrackets(std::string expression, int& index);


bool brackets(std::string expression, int& index)
{
	bool isExpression = false;
	if (expression[index] == '[')
	{
		index += 1;
		isExpression = isSquareBrackets(expression, index);
	}
```

```
	else if (expression[index] == '(')
	{
		index += 1;
		isExpression = isRoundBrackets(expression, index);
	}
	else if (expression[index] == '{')
	{
		index += 1;
		isExpression = isFiguredBrackets(expression, index);
	}
	return isExpression;
}

bool isSquareBrackets(std::string expression, int& index)
{
	bool b = false;
	if (expression[index] == '(') {
		index += 1;
		b = isRoundBrackets(expression, index);
		if (b == true) {
			if (expression[index] == '{') {
				index += 1;
				b = isFiguredBrackets(expression, index);
				if (expression[index] == ']')
				{
					b = true;
				}
				else b = false;
			}
			else b = false;
```

```
                index += 1;

            }
        }
        else if (expression[index] == '+') {
            b = true;
            index += 1;
            if (expression[index] != ']') {
                b = false;
            }
            index += 1;
        }
        return b;
}

bool isRoundBrackets(std::string expression, int& index)
{
        bool b = false;
        if (expression[index] == '{') {
            index += 1;
            b = isFiguredBrackets(expression, index);
            if (b == true) {
                if (expression[index] == '[') {
                    index += 1;
                    b = isSquareBrackets(expression, index);
                    if (expression[index] == ')')
                    {
                        b = true;
                    }
                    else b = false;
```

```
                }
                else b = false;
                index += 1;

            }
        }
        else if (expression[index] == '-') {
            b = true;
            index += 1;
            if (expression[index] != ')') {
                b = false;
            }
            index += 1;
        }
        return b;
}

bool isFiguredBrackets(std::string expression, int& index)
{
        bool b = false;
        if (expression[index] == '[') {
            index += 1;
            b = isSquareBrackets(expression, index);
            if (b == true) {
                if (expression[index] == '(') {
                    index += 1;
                    b = isRoundBrackets(expression, index);
                    if (expression[index] == '}')
                    {
                        b = true;
```

```
                        }
                        else b = false;
                }
                else b = false;
                index += 1;
        }
    }
    else if (expression[index] == '0') {
        b = true;
        index += 1;
        if (expression[index] != '}') {
            b = false;
        }
        index += 1;
    }
    return b;
}

int main()
{
    int index = 0;
    std::string expression;
    std::string filePath;

    std::cout << "Enter fileIn path: ";
    std::cin >> filePath;

    std::ifstream fileIn(filePath);

    if (!fileIn)
```

```
{
	std::cout << "File cannot be opened!\n";
	return 1;
}

int offset = 0;
while (getline(fileIn, expression))
{
	if (offset < expression.length())
		offset = expression.length();
}
offset += 5;

fileIn.clear();
fileIn.seekg(0);

std::cout << "Enter fileOut Path: ";
std::cin >> filePath;

std::ofstream fileOut(filePath);

while (getline(fileIn, expression))
{
	if (!brackets(expression, index) || expression.length() != index)
	{
		fileOut << std::endl << std::setw(offset) << std::left << expression
<< std::setw(offset) << std::left << "No";
		//std::cout << " = No\n";
		std::cout << std::endl << std::setw(offset) << std::left <<
expression << std::setw(offset) << std::left << "No";
```

```
                }
                else
                {
                        fileOut << std::endl << std::setw(offset) << std::left << expression
<< std::setw(offset) << std::left << "Yes";
                        //std::cout << " = Yes\n";
                        std::cout << std::endl <<std::setw(offset) << std::left <<
expression << std::setw(offset) << std::left << "Yes";
                }


                expression.clear();
                index = 0;
        }

        fileIn.close();
        fileOut.close();

        return 0;
}
```