МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МОЭВМ

ОТЧЕТ

по лабораторной работе №3

по дисциплине «Алгоритмы и структуры данных»

Тема: СТЕК И ОЧЕРЕДЬ

Вариант № 6-д

Студент гр. 8304 Рыжиков А. В.
Преподаватель Фирсов К. В.

Санкт-Петербург
2019

1 Цель работы.

Дано выражение. Необходимо проверить его синтаксическую корректность. Для решения данной задачи использовать стек на списках.

6. Проверить, является ли содержимое заданного текстового файла F правильной записью формулы следующего вида:

< формула > ::= < терм > | < терм > + < формула > | < терм > - < формула >

< терм > ::= < имя > | (< формула >) | [< формула >] | { < формула > }

< имя > ::= x | y | z

2 Описание программы

Программа решает поставленную задачу при помощи стека на списках. За основу для решения задачи создаём структуру Node и класс Stack. Выполнив операции над стеком, убедимся, что выражение принадлежит к синтаксически верному, выведем положительный результат, иначе отрицательный.

3.1 Зависимости и объявление функций, структуры данных

```
#include <iostream>
#include <memory>
#include <fstream>


struct Node {

    char data;
    std::shared_ptr<Node> next;
    std::shared_ptr<Node> prev;

    Node() {
        data = ' ';
        next = (nullptr);
        prev = (nullptr);
    }
};

class Stack {
    int count;
    std::shared_ptr<Node> lastElement;
public:
```

```
    Stack() {
        count = 0;
        lastElement = std::make_shared<Node>();;
    }


    void push(char i) {
        count++;
        lastElement->next = std::make_shared<Node>();
        lastElement->next->data = i;
        lastElement->next->prev = lastElement;
        lastElement = lastElement->next;
    }

    void pop() {
        if (count > 0) {
            count--;
            lastElement = lastElement->prev;
        }
    }

    int getCountStack() {
        return count;
    }

    char top() {
        return lastElement->data;
    }

};
```

Описание структур данных.

Структура Node содержит указатель на следующий и предыдущий элемент, хранит в себе char data.

Класс Stack содержит int count количество элементов, и std::shared_ptr<Node> lastElement верхний элемент в стеке и реализует основные функции стека.

3.2 Функция main.

Программа решает поставленную задачу при помощи рекурсии и иерархических структур, чтение происходит из файла test3.txt (также возможен ввод данных вручную) .

```
int main() {

    int your_choose = 0;

    std::cout << "If you want to enter data from a file, enter \'1\'\n";
    std::cout << "If you want to enter data manually, enter \'2\'\n";

    std::cin >> your_choose;

    if (your_choose == 1) {
        std::ifstream fin;
        fin.open("C:\\Users\\Alex\\Desktop\\test3.txt");

        if (fin.is_open()) {
            std::cout << "Reading from file:" << "\n";

            int super_count = 0;

            while (!fin.eof()) {

                super_count++;

                std::string str;
                getline(fin, str);

                std::cout << "test #" << super_count << " \"" + str + "\"" << "\n";
                mainCheck(str);

            }
        } else {
            std::cout << "File not opened";
        }

        fin.close();
    } else {
        if (your_choose == 2) {
            std::cout << "Enter data \n";
            std::string str;
            std::cin >> str;
            mainCheck(str);
        }
    }

    return 0;
}
```

Функция mainCheck.

Функция проверяет являются ли поданные данные синтаксически корректными, а также упрощает данные путём замены нескольких идентичных символов одним.

```
void mainCheck(std::string &name) {


    if (checkExtraneousSigns(name)) {
```

```
        for (char &i : name) {
            if (i == '{' || i == '[') {
                i = '(';
            }
            if (i == '}' || i == ']') {
                i = ')';
            }
            if (i == 'y' || i == 'z') {
                i = 'x';
            }
            if (i == '-') {
                i = '+';
            }
        }

        //std::cout << name << '\n';

        std::cout << checkExpression(name) << '\n';


    } else {
        std::cout << "extraneous signs or empty string" << '\n';
        std::cout << "0" << '\n';
    }

}
```

3.3 Вспомогательные Функции

1) bool checkExtraneousSigns(std::string &name)

Функция проверяет входящую строку на наличие посторонних символов.

```
bool checkExtraneousSigns(std::string &name) {

    for (char j : name) {
        if (j != '{' && j != '}' && j != '[' && j != ']' && j != '(' && j != ')' && j
!= 'x' && j != 'y' && j != 'z' &&
            j != '+' && j != '-') {
            return false;
        }
    }

    return name.length() != 0;

}
```

2) bool checkExpression(std::string &name)

В данной функции реализована основная логика. Создаётся стек. Стек заполняется данными, но при встрече закрывающих скобок, происходит проверка подстроки (все элементы от открывающей строки до закрывающей). В положительном случае вместо подстроки в стек добавляется заменяющая подстроку переменная x. В обратном случае возвращается false.

```
bool checkExpression(std::string &name) {

    auto stack = std::make_shared<Stack>();

    for (char i :name) {
        if (i == '(' || i == 'x' || i == '+') {
            stack->push(i);
        } else {

            if (i == ')') {
                std::string newString;
                while (true) {
                    char topElement = stack->top();
                    stack->pop();
                    if (topElement == '(' || stack->getCountStack() == 0) {
                        if (!checkSubstring(newString)) {
                            return false;
                        }
                        stack->push('x');
                        break;
                    }
                    newString += topElement;
                }
            }

        }
    }

    if (stack->getCountStack() != 0) {
        std::string newString;
        while (stack->getCountStack() != 0) {
            char topElement = stack->top();
            newString += topElement;
            stack->pop();
        }

        if (!checkSubstring(newString)) {
            return false;
        }


    }


    return stack->getCountStack() == 0;

}
```

3) bool checkSubstring(std::string name)

Функция проверяет подстроку на необходимое соответствие.

Правильная подстрока x+x+x+x+… т.д.

```
bool checkSubstring(std::string name) {

    int count = name.length();

    for (int i = 0; i < count; i++) {
        if (i % 2 == 0) {
            if (name[i] != 'x') {
                return false;
            }
        } else {
            if (i % 2 == 1) {
                if (name[i] != '+') {
                    return false;
                }
            }
        }
    }
    if (count == 0) {
        return false;
    }
    return count % 2 != 0;

}
```

4 Тесты

Тесты удовлетворяющие понятию формула

x+x

x+y

x+y-z

[{x}]+((y))+(((z)))

(x+{(y)+z})

{x+((y))+[z]}

x+y+z+y

{(x-y)+(z)}+[y]

[({({(x-y)+(z)}+[y])})]

x-{z+[y-{[z]+(x)}]}

x-{z+[y-{[z]+(x)}]}

Тесты не удовлетворяющие понятию формула

a+x

z+z+z*z

[{x}+((y))+(((z)))

(x+{(y)&z})

{x+((y))+[z]-}

{(x-y)+(z)}-+[y]

Вывод: *был реализован стеки, и на его основе решена поставленная задача. Изучена работа стека на списках. Были*

написаны тесты и проверена работоспособность программы.

Приложение

Код программы lab3.cpp

```
#include <iostream>
#include <memory>
#include <fstream>

struct Node {

    char data;
    std::shared_ptr<Node> next;
    std::shared_ptr<Node> prev;

    Node() {
        data = ' ';
        next = (nullptr);
        prev = (nullptr);
    }
};

class Stack {
    int count;
    std::shared_ptr<Node> lastElement;
public:

    Stack() {
        count = 0;
        lastElement = std::make_shared<Node>();;
    }


    void push(char i) {
        count++;
        lastElement->next = std::make_shared<Node>();
        lastElement->next->data = i;
        lastElement->next->prev = lastElement;
        lastElement = lastElement->next;
    }

    void pop() {
        if (count > 0) {
            count--;
            lastElement = lastElement->prev;
        }
    }

    int getCountStack() {
        return count;
    }
```

```
    char top() {
        return lastElement->data;
    }

};

bool checkExtraneousSigns(std::string &name) {

    for (char j : name) {
        if (j != '{' && j != '}' && j != '[' && j != ']' && j != '(' && j != ')' && j
!= 'x' && j != 'y' && j != 'z' &&
            j != '+' && j != '-') {
            return false;
        }
    }

    return name.length() != 0;

}


bool checkSubstring(std::string name) {

    int count = name.length();

    for (int i = 0; i < count; i++) {
        if (i % 2 == 0) {
            if (name[i] != 'x') {
                return false;
            }
        } else {
            if (i % 2 == 1) {
                if (name[i] != '+') {
                    return false;
                }
            }
        }
    }
    if (count == 0) {
        return false;
    }
    return count % 2 != 0;

}

bool checkExpression(std::string &name) {

    auto stack = std::make_shared<Stack>();

    for (char i :name) {
        if (i == '(' || i == 'x' || i == '+') {
            stack->push(i);
        } else {

            if (i == ')') {
                std::string newString;
                while (true) {
                    char topElement = stack->top();
                    stack->pop();
                    if (topElement == '(' || stack->getCountStack() == 0) {
```

```
                        if (!checkSubstring(newString)) {
                            return false;
                        }
                        stack->push('x');
                        break;
                    }
                    newString += topElement;
                }
            }

        }
    }

    if (stack->getCountStack() != 0) {
        std::string newString;
        while (stack->getCountStack() != 0) {
            char topElement = stack->top();
            newString += topElement;
            stack->pop();
        }

        if (!checkSubstring(newString)) {
            return false;
        }


    }


    return stack->getCountStack() == 0;

}


void mainCheck(std::string &name) {


    if (checkExtraneousSigns(name)) {


        for (char &i : name) {
            if (i == '{' || i == '[') {
                i = '(';
            }
            if (i == '}' || i == ']') {
                i = ')';
            }
            if (i == 'y' || i == 'z') {
                i = 'x';
            }
            if (i == '-') {
                i = '+';
            }
        }

        //std::cout << name << '\n';

        std::cout << checkExpression(name) << '\n';


    } else {
```

```
        std::cout << "extraneous signs or empty string" << '\n';
        std::cout << "0" << '\n';
    }

}

int main() {

    int your_choose = 0;

    std::cout << "If you want to enter data from a file, enter \'1\'\n";
    std::cout << "If you want to enter data manually, enter \'2\'\n";

    std::cin >> your_choose;

    if (your_choose == 1) {
        std::ifstream fin;
        fin.open("C:\\Users\\Alex\\Desktop\\test3.txt");

        if (fin.is_open()) {
            std::cout << "Reading from file:" << "\n";

            int super_count = 0;

            while (!fin.eof()) {

                super_count++;

                std::string str;
                getline(fin, str);

                std::cout << "test #" << super_count << " \"" + str + "\"" << "\n";
                mainCheck(str);

            }
        } else {
            std::cout << "File not opened";
        }

        fin.close();
    } else {
        if (your_choose == 2) {
            std::cout << "Enter data \n";
            std::string str;
            std::cin >> str;
            mainCheck(str);
        }
    }

    return 0;
}
```