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

ОТЧЕТ

по лабораторной работе №3
по дисциплине «Алгоритмы и структуры данных»
Тема: Линейные структуры данных: стек, очередь, дек
Вариант 5-д

Студент гр. 8304 _____ Чешуин Д. И.

Преподаватель _____ Фирсов М. А.

Санкт-Петербург
2019

Цель работы.

Познакомиться с часто используемыми на практике линейными структурами данных, обеспечивающими доступ к элементам последовательности только через её начало и конец, и способами реализации этих структур, освоить на практике использование стека, очереди и дека для решения задач.

Постановка задачи.

Правильная скобочная конструкция с тремя видами скобок определяется как:

< текст > ::= < пусто > | < элемент > < текст >

< элемент > ::= < символ > | (< текст >) | [< текст >] | { < текст > }

где < символ > - любой символ, кроме (,) , [,] , { , }. Проверить, является ли текст, содержащийся в заданном файле F, правильной скобочной конструкцией; если нет, то указать номер ошибочной позиции.

Описание алгоритма.

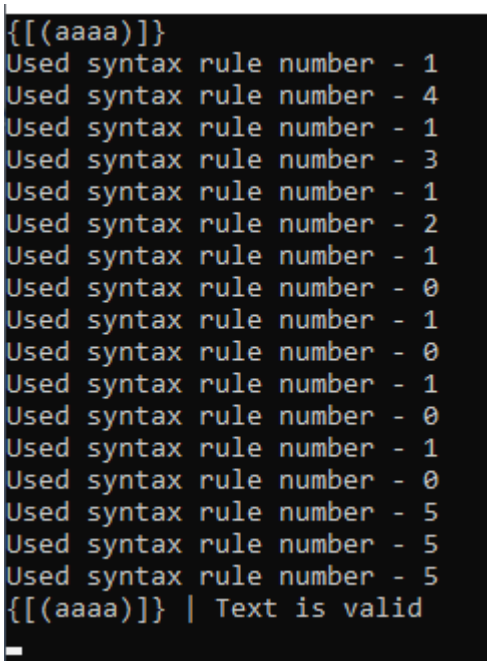
Программа считывает данные и отправляет их на проверку: в стек заносится форма скобочной конструкции, затем символы считываются по 1 и сравниваются с тем, что находится на вершине стека. Если они равны – из стека извлекается символ, если же нет, то ищется правило с данными символами. При его существовании символ на вершине стека заменяется на это правило, а при отсутствии – возвращается номер текущего символа, как адрес ошибки. Всё повторяется до тех пор, пока не будут считаны все символы и стек не опустеет.

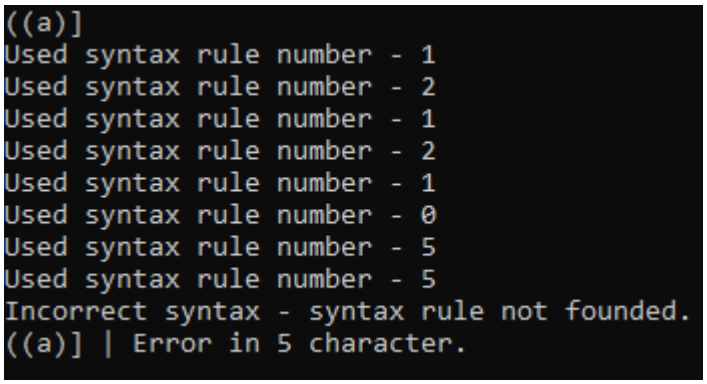
Спецификация программы.

Программа предназначена для валидации скобочной конструкции.

Программа написана на языке C++ с использованием CLI. Входными данными являются любые строки, введённые из консоли, либо из файла, переданного в качестве аргумента командной строки.

Рисунок 1- Результат работы программы





Тестирование.

Таблица 1 – Результаты тестирования программы

Input1	**Output**
a	Text is valid
(a)	Text is valid
(a){b}[c]	Text is valid
a(abc{def})b	Text is valid
()	Text is valid
({})	Error in 3 character.
(aaaaa(bbvbdfdf)	Error in 17 character.
)	Error in 1 character.

Анализ алгоритма.

Алгоритм работает за линейное время от размера строки.

Описание функций и СД.

Класс **DynamicStack** реализует структуру стека, а также методы для работы с ним.

Стандартные методы для работы со стеком:

```
T pop();
void void push(const T& data);
bool isEmpty();
T onTop();
```

Класс **Validator** реализует алгоритм проверки строки на корректность.

Статический метод для проверки строки на соответствие скобочной конструкции.

```
int check(std::istream& stream);
```

Принимает на вход ссылку на поток ввода, возвращает -1, если исходное выражение корректно и номер символа с ошибкой в случае ошибки. Строка анализируется посимвольно.

Выводы.

В ходе работы были приобретены навыки работы со стеком, изучены методы работы с ним (объявлять, заносить в него переменных и забирать их). Был изучен алгоритм анализа – LL-parser и на его основе реализован алгоритм левосторонней проверки выражения.

Приложение А. Исходный код программы.

dynamicstack.h

```
#ifndef DYNAMICSTACK_H
#define DYNAMICSTACK_H

#include<memory>
#include <iostream>
#include"node.h"

template <typename T>
class DynamicStack
{
public:
    typedef std::shared_ptr<DynamicStack> DynamicStackP;

private:
    typename Node<T>::NodeP head_ = nullptr;
    typename Node<T>::NodeP tail_ = nullptr;
    int elementsCount_ = 0;

public:
    DynamicStack() = default;
    DynamicStack(std::initializer_list<T> init);

    void push(const T& data);
    void push(const DynamicStack& stack);

    void pushBack(const T& data);
    void pushBack(const DynamicStack& stack);

    T pop();
    T onTop();

    size_t size();
    bool isEmpty();
    void clear();
};

template <typename T>
```

```
DynamicStack<T>::DynamicStack(std::initializer_list<T> init)
{
    for(auto value = init.begin(); value != init.end(); value++)
    {
        pushBack(*value);
    }
}

template <typename T>
void DynamicStack<T>::push(const T& data)
{
    typename Node<T>::NodeP newNode(new Node<T>);
    newNode->setData(data);

    if(head_ == nullptr)
    {
        head_ = newNode;
        tail_ = newNode;
    }
    else
    {
        newNode->setNext(head_);
        head_->setPrev(newNode);
        head_ = newNode;
    }

    elementsCount_ += 1;
}

template <typename T>
void DynamicStack<T>::push(const DynamicStack& stack)
{
    typename Node<T>::NodeP buf = stack.tail_;
    while(buf != nullptr)
    {
        T dataBuf = buf->data();

        push(dataBuf);

        buf = buf->prev();
    }
}

template <typename T>
void DynamicStack<T>::pushBack(const T& data)
{
    typename Node<T>::NodeP newNode(new Node<T>);
    newNode->setData(data);

    if(head_ == nullptr)
    {
        head_ = newNode;
        tail_ = newNode;
    }
    else
    {
        newNode->setPrev(tail_);
        tail_->setNext(newNode);
        tail_ = newNode;
    }

    elementsCount_ += 1;
}
```

```
template <typename T>
void DynamicStack<T>::pushBack(const DynamicStack& stack)
{
    typename Node<T>::NodeP buf = stack.head_;
    while(buf != nullptr)
    {
        T dataBuf = buf->data();

        pushBack(dataBuf);

        buf = buf->next();
    }
}

template <typename T>
T DynamicStack<T>::pop()
{

    typename Node<T>::NodeP buf  = head_;

    head_ = buf->next();

    buf->setNext(nullptr);

    elementsCount_ -= 1;

    if(head_ == nullptr)
    {
        tail_ = nullptr;
    }

    return buf->data();
}

template <typename T>
T DynamicStack<T>::onTop()
{
    return head_->data();
}

template <typename T>
bool DynamicStack<T>::isEmpty()
{
    return (elementsCount_ == 0);
}

template <typename T>
void DynamicStack<T>::clear()
{
    head_ = nullptr;
    tail_ = nullptr;
    elementsCount_ = 0;
}

template <typename T>
size_t DynamicStack<T>::size()
{
    return elementsCount_;
}

#endif // DYNAMICSTACK_H
```

iomanager.h

```
#ifndef CLIHANDLER_H
#define CLIHANDLER_H

#include <iostream>
#include <fstream>
#include <memory>
#include <sstream>

class IoManager
{
private:
    int argc_ = 0;
    char** argv_ = nullptr;
    int curArgNum_ = 1;

    std::istream* curInStream_ = nullptr;
    std::ostream* curOutStream_ = nullptr;

    void openNextStream();
public:
    typedef std::shared_ptr<IoManager> IoManagerP;

    IoManager(int argc, char** argv);
    ~IoManager();
    std::istream* nextStream();
    void writeLine(std::string line);
};

#endif // CLIHANDLER_H
```

node.h

```
#ifndef NODE_H
#define NODE_H

#include<memory>

template <typename T>
class Node
{
public:
    typedef std::shared_ptr<Node> NodeP;
    typedef std::weak_ptr<Node> NodeWP;
private:
    T data_;
    NodeWP prev_;
    NodeP next_ = nullptr;
public:
    Node() = default;

    void setData(T data);
    T data();

    void setPrev(NodeP node);
    NodeP prev();

    void setNext(NodeP node);
    NodeP next();
};

template<typename T>
void Node<T>::setData(T data)
{
    data_ = data;
}

template<typename T>
T Node<T>::data()
{
    return data_;
}

template<typename T>
void Node<T>::setPrev(NodeP node)
{
    prev_ = node;
```

```
}

template<typename T>
typename Node<T>::NodeP Node<T>::prev()
{
    if(prev_.expired())
    {
        return nullptr;
    }
    else
    {
        return NodeP(prev_);
    }
}

template<typename T>
void Node<T>::setNext(NodeP node)
{
    next_ = node;
}

template<typename T>
typename Node<T>::NodeP Node<T>::next()
{
    return next_;
}

#endif // NODE_H
```

validator.h

```
#ifndef VALIDATOR_H
#define VALIDATOR_H

#include <iostream>
#include "dynamicstack.h"

class Validator
{
private:
    enum Types // не enum class для удобства работы с таблицей валидации
    {
        CHARACTER,
        ROUND_OPEN, //открывающая круглая скобка
        QUAD_OPEN, //открывающая квадратная скобка
        FIGURE_OPEN, //открывающая фигурная скобка
        ROUND_CLOSE, //закрывающая круглая скобка
        QUAD_CLOSE, //закрывающая квадратная скобка
        FIGURE_CLOSE, //закрывающая фигурная скобка
        ELEMENT,
        TEXT
    };

    DynamicStack<Types> rulesTable[6] = {{CHARACTER},
                                         {ELEMENT, TEXT},
                                         {ROUND_OPEN, TEXT, ROUND_CLOSE},
                                         {QUAD_OPEN, TEXT, QUAD_CLOSE},
                                         {FIGURE_OPEN, TEXT, FIGURE_CLOSE},
                                         {}};

    int syntaxTable[9][9] = {{-1, -1, -1, -1, -1, -1, -1, -1, -1},
                             {-1, -1, -1, -1, -1, -1, -1, -1, -1},
                             {-1, -1, -1, -1, -1, -1, -1, -1, -1},
                             {-1, -1, -1, -1, -1, -1, -1, -1, -1},
                             {-1, -1, -1, -1, -1, -1, -1, -1, -1},
                             {-1, -1, -1, -1, -1, -1, -1, -1, -1},
                             {-1, -1, -1, -1, -1, -1, -1, -1, -1},
                             { 0,  2,  3,  4, -1, -1, -1, -1, -1},
                             { 1,  1,  1,  1,  5,  5,  5,  1, -1}};

    DynamicStack<Types> stack = {};

public:
    const static int VALID = -1;

private:
    Types typeOf(char elem);

public:
```

```
    Validator() = default;
    int check(std::istream& stream);
};
```

#endif // VALIDATOR_H

main.cpp

```
#include <iostream>
#include <string>
#include "ioManager.h"
#include "validator.h"


int main(int argc, char** argv)
{
    IoManager ioManager(argc, argv);
    Validator validator;
    std::string result;

    std::istream* stream = ioManager.nextStream();
    while(stream != nullptr)
    {
        std::getline(*stream, result);
        stream->seekg(0);

        int error = validator.check(*stream);

        if(error == Validator::VALID)
        {
            result +=  " | Text is valid";
        }
        else
        {
            result += " | Error in " + std::to_string(error) + " character.";
        }

        ioManager.writeLine(result);

        delete stream;
        stream = ioManager.nextStream();
    }

    return 0;
}
```

iomanager.cpp

```
#include "ioManager.h"

IoManager::IoManager(int argc, char** argv)
{
    argc_ = argc;
    argv_ = argv;

    if(argc_ < 2)
    {
        curInStream_ = &std::cin;
        curOutStream_ = &std::cout;
    }
}

void IoManager::openNextStream()
{
    if(curInStream_ == nullptr){
        curInStream_ = new std::ifstream();
        curOutStream_ = new std::ofstream();
    }

    if(curArgNum_ >= argc_)
    {
        if(curInStream_ != &std::cin)
        {
            delete curInStream_;
            delete curOutStream_;
        }

        curInStream_ = nullptr;
        curOutStream_ = nullptr;

        return;
```

```
    }

    std::ifstream* inFileStream = static_cast<std::ifstream*>(curInStream_);
    if(inFileStream->is_open())
    {
        inFileStream->close();
    }
    std::ofstream* outFileStream = static_cast<std::ofstream*>(curOutStream_);
    if(outFileStream->is_open())
    {
        outFileStream->close();
    }

    while(curArgNum_ < argc_ && !inFileStream->is_open())
    {
        std::cout << "Try to open file - ";
        std::cout << argv_[curArgNum_] << std::endl;

        inFileStream->open(argv_[curArgNum_]);

        if(inFileStream->is_open())
        {
            std::string outFile(argv_[curArgNum_]);
            outFile += " - results";
            outFileStream->open(outFile);

            std::cout << "File opened." << std::endl << std::endl;
        }
        else
        {
            std::cout << "Can't open - file not founded" << std::endl << std::endl;
        }

        curArgNum_ += 1;
    }
}

std::istream* IoManager::nextStream()
{
    if(curInStream_ == nullptr)
    {
        openNextStream();

        if(curInStream_ == nullptr)
        {
            return nullptr;
        }
    }

    while(curInStream_->peek() == EOF)
    {
        openNextStream();

        if(curInStream_ == nullptr)
        {
            return nullptr;
        }
    }

    std::string buffer;

    std::getline(*curInStream_, buffer);
    if(buffer == "")
    {
        return nullptr;
    }

    std::stringstream* sstream = new std::stringstream();
    *sstream << buffer;

    return sstream;
}

void IoManager::writeLine(std::string line)
{
    *curOutStream_ << line << std::endl;
}

IoManager::~IoManager()
```

```
{
    if(curInStream_ != nullptr && curInStream_ != &std::cin)
    {
        delete  curInStream_;
        delete  curOutStream_;
    }
}
```

validator.cpp

```
#include "validator.h"

Validator::Types Validator::typeOf(char character)
{
    if(character == '(')
    {
        return ROUND_OPEN;
    }
    if(character == '[')
    {
        return QUAD_OPEN;
    }
    if(character == '{')
    {
        return FIGURE_OPEN;
    }
    if(character == ')')
    {
        return ROUND_CLOSE;
    }
    if(character == ']')
    {
        return QUAD_CLOSE;
    }
    if(character == '}')
    {
        return FIGURE_CLOSE;
    }

    return CHARACTER;
}

int Validator::check(std::istream& stream)
{
    int position = 1;
    char nextChar = 0;

    stack.clear();
    stack.push(TEXT);

    for(; stream.peek() != EOF;)
    {
        if(stack.isEmpty())
        {
            std::cout << "Incorrect syntax - excess characters." << std::endl;

            return position;
        }

        nextChar = static_cast<char>(stream.peek());

        Types expectedType = stack.onTop();
        Types realType = typeOf(nextChar);

        if(expectedType == realType)
        {
            stream.get();
            stack.pop();

            position += 1;
        }
        else
        {
            int syntaxRule = syntaxTable[expectedType][realType];
            if(syntaxRule != -1)
            {
                stack.pop();
                stack.push(rulesTable[syntaxRule]);
```

```
                std::cout << "Used syntax rule number - " << syntaxRule << std::endl;
            }
            else
            {
                std::cout << "Incorrect syntax - syntax rule not founded." << std::endl;

                return position;
            }
        }
    }


    if(!stack.isEmpty())
    {
        if(stack.size() > 1 || stack.onTop() != TEXT)
        {
            std::cout << "Incorrect syntax - early end of string." << std::endl;

            return position;
        }
    }

    return VALID;
}
```